

决战Nginx



高性能Web服务器详解与运维

- 案例来自实际项目
- (proxy_cache、proxy_store、Memcached、Varnish和NCACHE)五大缓存详解与应用
- 40多个模块精解
- Nginx服务器高可用实现
- 使用PCRE实现强大的正则匹配
- Nginx实现灵活的域名配置

陶利军 编著

清華大學出版社



决战



高性能Web服务器详解与运维

陶利军　编著

清華大學出版社
北京

内 容 简 介

在这个点击率就是生命的时代，高可用是不可少的。本书完整讲述了Nginx服务器的各种技术细节以及安装、部署、运维等方面的内容。

本书第一部分首先讲述了Nginx服务器的功能、模块管理和进程管理，然后讲述Nginx如何处理请求，在这个基础之上再认识Nginx提供的服务器的名字，Nginx服务器最大的焦点在于高并发和反向代理，在不多却足够使用的模块下实现了更多的功能。

在第二部分中，通过具体使用实例讲述了Nginx的模块（包括官方模块和第三方模块），并详细介绍了充分使用Nginx的方式方法。同时在这里使用了Heartbeat服务实现Nginx服务器的高可用。

本书的最后一部分是关于Nginx使用缓存技术的方法，共列举了Nginx使用的五大缓存，特别是广泛使用的代理缓存、Memcached和Varnish，另外对于Memcached服务器的使用贯穿了整套书。在本书中着重讲述了它的协议、原理和使用，而在本书姊妹篇中则通过不同语言的客户端对Memcached服务器实现具体使用。

本书适用于广大的Linux爱好者、具有一定Linux基础的系统管理员、Linux下的Web服务器管理员、Linux服务器下动态语言开发人员、Nginx服务器管理员、培训中心师生、运维人员以及一切应该了解和使用Nginx的用户。

图书在版编目（CIP）数据

决战Nginx系统卷：高性能Web服务器详解与运维/陶利军编著.—北京：清华大学出版社，2012.6

ISBN 978-7-302-28784-1

Ⅰ.①决… Ⅱ.①陶… Ⅲ.①Web服务器 Ⅳ.①TP393.09

中国版本图书馆CIP数据核字（2012）第084716号

责任编辑：栾大成
封面设计：杨如林
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，　http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，　c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：188 mm×260 mm　　**印　张：**42.75　　**插页：**1　　**字　数：**1153千字
版　　次：2012年6月第1版　　**印　次：**2012年6月第1次印刷
印　　数：1～5000
定　　价：79.00元

产品编号：047048-01

前 言

Nginx 服务器在互联网推波助澜的作用下脱颖而出，创下了高并发的记录，因此，在短短 10 年的发展中，在全世界前 100 万的网站中，已经有 5.1%的网站使用了 Nginx 服务器，使得 Nginx 成为继 Apache（70.2%）和 IIS（20.5%）之后的第三大 Web 服务器软件，而且它的使用数量与日俱增，直逼 Apache 的市场。有人说，Nginx 将会取代 Apache 的市场，我们且不关心 Nginx 能不能取代 Apache， 1996 年 4 月以来，Apache 一直是 Internet 上最流行的 HTTP 伺服器，而且在 Linux 系统下 Apache 几乎是不二的选择，而在 2002 年诞生的 Nginx 服务器在这种形式下能够崛起那绝对是个奇迹，它有两个方面能够打败 Apache 服务器，一是高并发，二是节省资源，即轻量级。在我们现有的应用中，基本上将 Apache 跑在了后台，而是同 Nginx 服务器代理访问了。

使用 Nginx 服务器就是使用它的两大特点，一是高并发访问，二是代理，它能够快速地解析静态文件，而对于动态语言实现的动态程序则传递到后台的服务，实现了动静网页解析的分离。

Nginx 是一个自由的、开源的、高性能的 HTTP 服务器和反向代理，同时也是一个 IMAP/POP3 的代理服务器，它是由 Igor Sysoev 于 2002 年开发，并且在 2004 年发布了第一个版本，在互联网上使用 Nginx 的主机近乎 6.55% （13.5M）。

Nginx 之所以能够脱颖而出闻名世界，是因为它的高性能、稳定性、丰富的功能设置、简单的配置和资源消耗低。

Nginx 解决了服务器的 C10K 问题，它的设计不像传统的服务器使用线程处理请求，取而代之的是使用了一个更加高级的机制——事件驱动机制，而且是异步事件驱动结构。

即使你不希望处理成千上万的并发请求，你同样能够从 Nginx 的高性能和低消耗内存（占用内存小）的结构中获益。Nginx 的使用规模很全面：从很小的 VPS 到服务器集群都可以。

Nginx 强有力地用在了一些高知名度的站点，例如 WordPress、Hulu、Github、Ohloh、SourceForge 和 TorrentReactor。

本套书所包括的内容

本套书共包括卷 1 和卷 2 两本书，共 10 个部分（包括了 86 章的内容，其中第十部分是一个独立部分，没有章节），其中卷 1 包括了前 3 部分的内容，而卷 2 包括了后面的 7 部分内容。

卷 1 内容

第 1 部分　Nginx 服务器

第 2 部分　Nginx 服务器的功能模块

第 3 部分　Nginx 与缓存

第 1 部分　Nginx 服务器

该部分包括了服务器的功能、Nginx 服务器的模块管理和进程管理、5XX 错误处理、协助用户操作 Nginx 的工具和高可用的 Nginx 等内容。

第 1 章　Nginx 的功能

本章认识 Nginx 服务器的基本功能和其扩展功能，以及 Nginx 核心模块的相关指令和变量。

第 2 章　Nginx 的模块管理和进程管理

Nginx 同 Apache 一样，同样使用了模块化管理，但是与 Apache 有很大的不同，如果说 Apache 支持“热插拔”（就是说如果对 Apache 添加模块不用重新编译 Apache，而只是添加必要的模块，然后再重新载入 Apache 就可以了），那么 Nginx 则必须“重启动”，就是说如果要对 Nginx 服务器添加模块，那么需要重新编译 Nginx 才可以添加相应的功能模块，因此在这一点上要比 Apache 服务器麻烦。

第 3 章　Nginx 如何处理一个请求

Nginx 服务器在处理一个请求时是按照两部分例子进行的，第一部分是 IP、域名，第二部分是 URI。下面本章将分析这两部分是如何进行工作的。

第 4 章　服务器名字

服务器的名字是由指令 server_name 来定义，并且也决定了使用哪一个 server 区段来提供对客户端请求的响应。服务器名字的定义可以使用准确的名字（exact name）、通配符名字（wildcard name）或者是正则表达式。

第 5 章　协助用户操作 Nginx 的工具

工具 nginx.vim 是一个辅助工具，通过下面的配置，在使用时它将成为 vim 工具的一部分，具体的作用是用于编辑 Nginx 的配置文件。

这个工具是一个 shell（Bash）脚本，是 Debian 系统下用于控制 Apache2.2 虚拟主机命令 a2ensite 和 a2dissite 的复制版，用于控制 Nginx。原始的 a2ensite 和 a2dissite 是用 Perl 语言编写，a2dissite 是 a2ensite 的一个符号链接，在本工具中，开发者遵循了同样的方法，例如，nginx_dissite 是 nginx_ensite 的一个符号链接。

如果没有安装 Apache，那么可以使用该工具来产生和管理 htpasswd 文件。

Nginx 启动脚本，由于在默认的安装包中没有提供管理脚本，为了管理方便，我们需要为它添加一个启动/关闭/脚本。

第 6 章　5XX 错误处理

本章介绍了 500、502 和 504 错误代码的原因及处理。

第 7 章　使用 TCMalloc 优化 Nginx

TCMalloc 即 Thread-Caching Malloc 的缩写，它是由 Google 公司开发的一款开源工具 google-perftools 中的一成员。TCMalloc 在内存的分配上效率和速度要比标准的 glibc 库高得

多，它不但可以用来优化高并发下的 MySQL，从而可以降低系统的负载，同样它可以用于 Nginx 实现同样的功能，因此，对于高并发的 Nginx 来说无疑是如虎添翼。

第 8 章　PCRE 正则表达式

由于在使用 Nginx 的过程中会用到正则表达式，因此有必要对这一部分内容进行一个清晰的说明。并且讲述了 pcre-config 和 pcretest 命令，以及 Nginx 服务器支持 UTF8 正则表达式匹配。

第 9 章　Nginx 高可用的实现

我们在 Nginx 下实现高可用不是担心 Nginx 服务出问题，而是担心硬件的问题，因此在设计通过 Heartbeat 服务来提供高可用时没有通过 Heartbeat 服务器来控制 Nginx 服务器的启动、停止等等，而只是通过它来提供了一个“浮动”的 IP 地址，次 IP 就是 Nginx 服务器提供访问的 IP，也就是被解析到相应域名的 IP 地址。

第 10 章　10 个 QA

10 个经常被问的问题。

第二部分　Nginx 服务器的功能

在中篇中通过“讲”的方式，进行了 42 讲，详细的讲述了 Nginx 提供的模块实现的功能：

第 11 章　限制流量
第 12 章　限制用户并发连接数
第 13 章　修改或隐藏 Nginx 的版本号
第 14 章　配置 FLV 服务器
第 15 章　Nginx 的访问控制
第 16 章　提供 FTP 下载
第 17 章　Nginx 与编码
第 18 章　网页压缩传输
第 19 章　控制 Nginx 如何记录日志
第 20 章　map 模块的使用
第 21 章　Nginx 预防应用层 DDoS 攻击
第 22 章　为 Nginx 添加、清除或改写响应头
第 23 章　重写 URI
第 24 章　Nginx 与服务器端包含
第 25 章　Nginx 与 X-Sendfile
第 26 章　在 Nginx 的响应体之前或之后添加内容
第 27 章　Nginx 与访问者的地理信息
第 28 章　Nginx 的图像处理
第 29 章　location 中随机显示文件
第 30 章　后台 Nginx 服务器记录原始客户端的 IP 地址

第三部分　Nginx 与缓存

通过 Nginx 来实现缓存功能基本上有四类方法，其中第一类方法 Nginx 自带，即 proxy_cache、proxy_store 和 memcached，在没有 proxy_cache 之前只能用 proxy_store 缓存页面，但是 memcached 需要第三方软件 Memcached；第二类是通过添加第三方模块，下面我们通过例子来分析；第三类是通过 Varnish 服务器。这样共有五种方法来实现缓存。

第 53 章　缓存技术——proxy_cache

Nginx 的这个功能从 0.7.48 版本开始提供的，它开始支持类似 Squid 的缓存功能。它的原理是把 URL 及相关变量的组合当做 Key，再用 MD5 编码，编码后的哈希值作为文件名，然后再将缓存的文件保存在硬盘中。我们现在使用的是 0.8.53 版本，早在 0.8.31 版本中 proxy_cache 就比较完善了，之所以说它比较完善其实就是说它没有缓存清除机制，但是通过第三方模块 ngx_cache_purge 来清除指定的 URL 缓存，它支持任意 URL 链接、支持 404/301/302/200 状态码缓存，因此，在使用反向代理的同时使用 Nginx 的 proxy_cache 缓存机制是个很不错的做法。

第 54 章　缓存技术——proxy_store

在没有 proxy_cache 之前，都是使用 proxy_store，由于 Nginx 的 proxy_store 技术当时

在设计时没有采用任何刷新机制，因此，对于缓存的管理上也就更人为了一些（你不觉得这么说更好听些？！），我们可以自己写个脚本进行缓存清理等等。使用 proxy_store 的原理其实也很简单，那就是 Nginx 首先在本地查找客户端请求的内容，如果找不到就去 proxy_pass 指定的后端服务器上查找，然后会被保存到本地的缓存中。使用 proxy_store 技术有一个缺点，其实也能认为是优点，它会在本地缓存中构建一个和远程服务器——proxy_pass 指定的后端服务器目录结构完全一样的结构（真得可以叫镜像了！），这是从它的优点方面讲。从缺点方面讲，那么它会浪费很大的磁盘空间，如果没有一个足够大的硬盘或者是没有及时地清除缓存中的过时文件，那么将是一个可怕的问题，所以你要么准备一个足够大的硬盘，要么及时或是有计划地清除缓存。

第 55 章　缓存技术——Memcached

这是使用比较多的一种缓存方式，使用 Memcached 模块也会分为四部分分析，第一部分就是 Nginx 的 memcached 模块本身，而另一部分就是 Memcached 服务，第三部分就是 Nginx 的配置文件，将这二者结合起来，就是第四部分客户端。

第 56 章　缓存技术——NCACHE

这个缓存方式我们只需要了解一下就可以了。第三方模块用得较多的就是新浪网的开源项目——NCache。这是一个比较古老的模块了，NCache，Nginx Cache，它只支持 Nginx 的 0.6.x 版本，对于 Nginx 的其他版本并不支持，相反的是在后面的版本中是以 Nginx 内核形式出现的。

第 57 章　缓存技术——Varnish

在 Nginx 中 Varnish 缓存是通过代理模块来实现的，在我使用的 Varnish 缓存服务器中，其中之一就是为了 Apache 而使用它。

Varnish 是一个高性能的、先进的 Web 加速器，它可以安装在 Linux 2.6，FreeBSD 6/7 和 Solaris 10 系统上，以便发挥其高效的性能。

卷 2 内容

第 1 部分　Nginx 与 php

第 2 部分　Nginx 与 Python

第 3 部分　Nginx 与 Perl

第 4 部分　Nginx 与 Java

第 5 部分　Nginx 与 Ruby

第 6 部分　Nginx 与 ASP.NET

后记　Nginx 与 Apache

第 1 部分　Nginx 与 PHP

要将 Nginx 和 PHP 结合，让 Nginx 解析静态网页，而 PHP 的动态网页交给 PHP 处理。

解决方法从大的方向有两类：（从 Nginx 角度来讲）一类是使用 Nginx 的代理模块，而另外一类则是使用 FastCGI 模块；而从 PHP 角度来讲则是 FastCGI 进程，它的方法有三种：一种是以 php-fpm 方式运行，第二种是 PHP 自带的 fastcgi server，第三种就是借助 lighttpd 带的 spawn-fcgi（听起来有点龌龊，但是确实可行，有时候还必须使用这种方法）。

第 1 章　环境部署

本章作为该部分的第 1 章，我们首先来部署环境。在 PHP 方面我们使用了 php-fpm，而在 Nginx 方面我们使用的是 FastCGI 模块。

第 2 章　PHP 访问 Memcached

为理解 Memcached 的使用，我们在这里讲述两个例子，一个是微博网站的部署，具体就是用 Memcached 存储什么数据，我没有仔细分析过，只负责部署和测试是否符合要求就够了；另一个是在我所运维环境中使用的一个例子，非常简单且实用。

在 PHP 下使用 Memcached 服务器，有两个选择，一个是纯 PHP 框架开发的 memcache，而另一个则是使用 libmemcached 的 memcached。

第 3 章　php-fpm 的状态

了解 php-fpm 的工作状态。

第 2 部分　Nginx 与 Python

本部分我们将了解到 Nginx 的 uwsgi 模块、uWSGI 服务器以及 ngx_cache_purge，当然根据需要我们还可以对 Django 架构了解一下。

第 4 章　uWSGI 服务器

uWSGI 是一个快速的、自维护的、对开发者和系统管理者友好的应用程序容器，是纯 C 语言开发的服务器。

在它的诞生之日，uWSGI 只是作为一个 WSGI 服务器，但是随着时间的推移，它现在已经演变为一个完整的网络、集群 Web 应用服务器，可以执行消息、对象传递、缓存、RPC 和进程管理。

它使用的协议是 uwsgi（注意，所有的字母都是小写，该协议已被 Nginx 和 Cherokee 的发行版本所包含），所有网络或进程间通信均使用 uwsgi 协议。

uWSGI 可以运行在预 fork 模式、线程模式、异步模式等，并且支持 green threads、coroutines 各种形式，例如 uGreen，Greenlet，Stackless 和 Fiber。

对于管理人员来说，uWSGI 服务器提供了各种配置方法：命令行、环境变量、XML、ini、yaml、json、sqlite3 数据库和 LDAP。

除此之外，它的设计完全模块化，这意味着，可以使用不同的插件以便满足不同的技术应用，从而实现兼容性。

第 5 章　Nginx 的 uwsgi 模块

该模块能够使得 Nginx 与 uWSGI 进程进行交互，并且可以控制传递给 uWSGI 进程的参

数。对于 uwsgi 协议和 uWSGI 服务器，uWSGI 服务器就是 uwsgi 协议的一个实现。

第 6 章　环境部署

在了解了 Nginx 的 uwsgi 模块、uWSGI 服务器以及 ngx_cache_purge 之后，当然根据需要我们还可以对 Django 架构了解一下，在这里我们按照一个全新的环境来布置，即从安装 Python 开始。在后面的实例部分，我们使用了 Python 的 2.43 版本，这是 Red Hat 系统自带的，同时也使用 2.7.2 版本，具体的版本根据自己的需要去选择。

第 7 章　实例运行

下面我们通过实例的方式来讲述，在下面的内容中将会讲述 8 个运行实例。

- 实例 1：运行开发服务器
- 实例 2：以 uWSGI 方式运行
- 实例 3：使用 Django 框架
- 实例 4：一个 uWSGI 实例实现对多个虚拟主机的支持
- 实例 5：分别监听在不同端口上的两个 uWSGI 实例
- 实例 6：针对 Nginx uwsgi 模块应用举例的一个具体实现
- 实例 7：集群的实现
- 实例 8：会话存储（基于数据库的方式和基于 Memcached 的两种方式）

第 8 章　缓存

对于 Memcached 服务器，Python 客户端方式选择有三种：一是 python-memcached，它的下载地址是 http://www.tummy.com/Community/software/python-memcached/，最新版本为 1.47；二是 cmemcache，它的下载地址为 http://gijsbert.org/cmemcache/，最新版本为 0.95；三是 libmemcached，它的下载地址为 http://download.tangent.org/，最新版本为 0.9。

第 9 章　会话

session 是 Django 中的一个高级工具，它可以存储用户的个人信息，以便用户在下次访问该网站时使用这些信息，session 的基础还是 cookie，但是它能够提供一些更加高级的功能。

Django 提供了完全支持匿名会话的功能，它的会话结构让每个网站的访问者存储并且检索任意数据。它将数据存储在服务器端并且对发送和收到的 cookies 做摘要，cookies 包含一个会话 ID，而不是数据本身。Django 只在需要的时候才发送 cookie，如果我们没有设定任何的 session 数据，它不会送出 cookie。

此外，还需要明白一点，Django 的会话（session）框架是完全基于 cookie 的，并且它也只能是基于 cookie，而不会像其他一些软件（例如 PHP）那样，在 session 不能正常工作时，就会把 session ID 放到 URL 中。任何事情只要存在就有它的道理，这一决定是经过深思熟虑的，将 session ID 放到 URL 中的那种方法不仅使得 URL 很丑陋，并且 session ID 还有可能通过 Referer 头泄漏出去，从而给网站带来安全隐患，这就是 Django 基于 cookie 的原因。

第 3 部分　Nginx 与 Perl

在 Nginx 的设计上并不支持 CGI，这不是它本身的缺陷，而是一个重要的举措：因为 Nginx

不能够直接执行外部程序（CGI），因此，怀有恶意的人就不能随意地直接执行外部脚本程序。当然，什么事情都是有方法可循的，例如，PHP FastCGI 脚本的支持，当我们将一个 PHP 脚本上传到一个可以执行 PHP FastCGI 的目录中，那么该脚本就可以执行，这个我们已经了解过了，但是这种方法是有一点难度，但是相对来说比较安全。但是有时候我们也需要简单的 CGI 程序支持，我们将介绍一个简单的 CGI 来替代 FastCGI，我们称这样的程序为 CGI 程序，它是用 Perl 语言实现的。

本部分实现了三个应用即 CGI、perl-FCGI 和 Nginx 内置的 Perl 模块。

第 10 章　Nginx 提供 Perl CGI 访问

第 11 章　Nginx 与 perl FastCGI

要将 Nginx 和 Perl 结合，让 Nginx 解析静态网页，而 Perl 的动态网页交给 Perl 处理。解决方法从大的方向有两类：（从 Nginx 角度来讲）一类是使用 Nginx 的代理模块，而另外一类则是使用 FastCGI 模块，而从 Perl 角度来讲则是 FastCGI 进程。

有关 Memcached 服务器的使用通过以下客户端作为实现，Perl 的 memcached 客户端有：

- Cache::Memcached
- Cache::Memcached::Fast
- Memcached::libmemcached
- Cache::Memcached::libmemcached

第 12 章　Nginx 通过内置的 Perl 模块执行 Perl 程序

通过使用该模块，Nginx 服务器可以直接在 Nginx 内部执行 Perl，或者是通过 SSI 来调用 Perl。

第 4 部分 Nginx 与 Java

在 Java 部分我们选择了 Tomcat 服务器作为 Java 的解析器。在这里着重分析了 Tomcat 的配置文件，及其实际应用中的配置。

第 13 章　环境部署

在我们实际使用 Nginx 时，往往不是 Nginx 单独工作，相反的是，它总是与一些动态语言所使用的“解析器”（这个名字是我起的，不是权威，可不要效仿！）构成动态网站，例如，我们将要接触的 Tomcat。Tomcat 在与 Nginx 的搭配中，我们使用 Nginx 的反向代理功能。

第 14 章　Nginx 与 Tomcat 的结合

Nginx 与 Java 的实现方式是通过代理模块来实现的，在 Nginx 方面使用代理模块，而 Java 方面，可以使用 Tomcat 也可以使用 Resin，还可以是其他的应用服务器。因此在这里我们分为三方面来讲述这个问题：一是代理模块、二是 Tomcat 或 Resin 的配置，三是这两者的结合。将 Nginx 作为前台服务器而把 Tomcat 作为后台服务器，由 Nginx 解析静态文件，而将动态的 JSP 网页发送到后台的 Tomcat 服务器。Nginx 在默认时就将该代理（Proxy）模块建立在内，通过使用该模块允许你将客户端的 HTTP 请求转发到后台服务器。

第 15 章　配置 server.xml 文件

从本章开始我们将来认识 Tomcat 的配置文件。对于我们运维人员来说，服务器的精髓都在配置文件中。

第 16 章　配置 web.xml 文件

首先我们要确定的是这个文件会有两种位置：一种是在$CATALINA_BASE/conf/目录下，而另一种情况是在$CATALINA_BASE/webapps/[webapp]/WEB-INF/目录下，前者可以叫做全局 web.xml 配置，而后者是各个 Web 应用程序自己的配置，Tomcat 在部署 Web 应用程序时，可能会包括两种情况，一种是 Tomcat 在启动时，第二种是 Tomcat 对应用程序进行重新加载时。在这两种情况下，Tomcat 都会先去读取全局配置（即 conf/web.xml），然后再去读取各自 web 应用程序目录下的配置（即 WEB-INF/web.xml）。

但是我们需要明白两点：一是全局配置（conf/web.xml）会将配置文件中的配置应用到所有的 Web 应用程序，而每个 Web 应用程序自己的配置只能应用自己，因此不要在全局配置文件中配置基于特定应用程序的资源；二是对于每个 Web 应用程序来说，它既可以有自己的 web.xml，也可以没有，如果没有该配置文件，Tomcat 会给出提示，但不会停止部署该应用程序。

第 17 章　配置 context.xml 文件

本章我们来了解 context.xml 文件的配置情况。

第 18 章　配置 tomcat-users.xml 文件

本章将要了解的是 Tomcat 的用户配置文件，包括用户、角色和密码。

第 19 章 配置 catalina.policy 文件

对于本文件来说，最大的特点是只有开启了 Security Manager 后，该文件才被使用，如果没有开启，那么这个配置就是一个摆设。

这一部分内容严格地来说属于 Java 的安全，和 Tomcat 本身关系并不大，但是对于我们运维来说，真不知道什么是不需要的，老实说这一部分内容如果处理不好，一是可能会引发安全问题，二是可能会导致软件工程师所写的 Java 程序在 Tomcat 上运行不了。

第 20 章 配置 catalina.properties 文件

在该部分内容中讲述两个内容：一是 catalina.properties 文件分析，二是 Loader 元素和类的加载器。

虽然两者看起来是不同的内容，但是在一定程度上，它们确实完成相同的工作。

第 21 章 在容器元素中可以使用的过滤器

Tomcat 提供了许多过滤器（Filters），可以将这些过滤器配置在所有 Web 应用程序都可以使用的$CATALINA_BASE/conf/web.xml 文件中，也可以配置在单独的 WEB-INF/web.xml 文件中，以便应用到相应的 Web 应用程序中。

第 5 部分　Nginx 和 Ruby

本部分的内容为：Ruby、Rails，即 RoR 和 Passenger。

首先讲述了 Ruby 的安装及相关工具 gem，然后是 Passenger，Passenger 与 Nginx 的结合安装方式有两种，即 Passenger 安装和 Nginx 模块式安装，以及 Passenger 提供的相关分析和系统维护工具。Rails 架构是一个纯 Ruby 的 MVC 架构，我们在这部分从运维的角度详细地分析了 Rails，包括 Rails 的相关技术和项目实例分析。

Rails 提供了缓存技术，在分析 Rails 提供的缓存技术基础之上我们着重使用了 Dalli 缓存接口和 Memcached 服务器。

第 22 章　环境部署

本章是该部分的第 1 章，因此我们首先要部署 Ruby 环境，Nginx 与 Python 的结合是通过 Passenger 来实现的，因此在本章中要介绍两种部署 Passenger 的方法，Passenger 模块及其在 Nginx 中的配置。

第 23 章　走进 Rails

在这一章中我们将认识 Ruby 的一个框架 Rails，也就是 RoR 的另一个 R。

第 24 章　缓存

Rails 架构提供了不同的缓存技术，对于行为（action）和片段（fragment）缓存策略来说可以使用这些缓存技术，而对于页面（page）缓存技术来说，它总是存储在磁盘上。

缓存技术有以下几种：

- 内存缓存技术
- 文件系统缓存技术
- Memcached 服务器技术
- Ehcache 缓存技术
- Dalli —— Memcached 的客户端

第 6 部分　Nginx 与 ASP.NET

本部分讲述的是通过 Mono 将 ASP.NET 程序运行在 Linux 操作系统下，ASP.NET 跑在 Linux 系统下的使用实例不是没有，但也不是很多，好像是越来越多的出现这种跨平台的移植现象。

第 25 章　Mono

在本章中我们简单地认识一下 Mono。

在本章需要了解两个问题，一个就是什么是 Mono，另一个是 Mono 的使用级别。

第 26 章　Nginx 与 ASP.NET 的解决方案

在本章我们实施了三个方案，每一个方案都有自己的特点，在具体的实施中可以做适当的选择。在这部分中讲述了三种结合方案：

- 方案一：Nginx+mono+ fastcgi-mono-server

- 方案二：Nginx+mono+Jexus
- 方案三：Nginx+mono+xsp

第 27 章　Session 存储

由于 HTTP 协议是无状态的协议，因此在客户端每次访问 Web 页面时，作为服务器端都要重新打开会话，作为开发人员，或者说是从用户使用的角度来看待这种无状态的协议是不会自动维护客户端所访问的环境信息，因此需要 session。

第 28 章　缓存

关于缓存我们已经讨论了很多，从缓存存储的位置来讲，归纳起来无非三种：

- 服务器端缓存
- 第三方缓存
- 客户端缓存

从缓存内容来讲，分为两种：

- 缓存动态内容
- 缓存静态内容

对于静态文件的缓存，由于我们通过 Nginx 做了动静分离，因此，通常静态文件是通过 Nginx 提供服务。

从缓存由动态程序产生的缓存来看，又分为以下类型：

- 全部内容缓存
- 部分内容缓存

那么我们在这里作为总结分析一下。

第 29 章　Nginx 代理 IIS

与 Mono 相比，更多的 ASP.NET 是运行在 IIS 服务器之下。在一个大型的网站中，往往会出现使用多种语言的情况。如果使用了 ASP.NET 技术，就免不了有 Windows 的系统，也就是使用了 IIS。在这种情况下，为了使用 Nginx 的高效性（运行在 Linux 下），可以使用 Nginx 的代理模块来实现，将 IIS 作为后台服务器来运行，而让 Nginx 服务器运行在前台

后记　Nginx 与 Apache

这是一个独立的部分，不再有章节，一是为了怀念 Apache，二是为了更好地使用 Apache，在没有 Nginx 的那个年代，Apache 为我们的网站作出了不可磨灭的贡献！

Nginx 服务器的功能再强大，不要忘记为我们立下汗马功劳的 Apache 服务器，否则你就是一个忘恩负义的人。

我不赞成用 Nginx 替换掉 Apache，相反我们可以使用混合的环境，对于这种 Nginx、Apache 混合使用的架构中，一是根据实际的应用来逐步使用 Nginx，二是在现有基础上实现 Nginx+Apache，具体的方法还是反向代理，让 Apache 运行在后台，而 Nginx 运行在前台。

使 用 对 象

- 广大的 Linux 爱好者。
- 具有一定 Linux 基础的系统管理员。
- Linux 下的 Web 服务器管理员。
- Linux 服务器下动态语言开发人员。
- Nginx 服务器管理员。
- 培训中心。
- 运维人员。
- 一切应该了解和使用 Nginx 的用户。

内 容 声 明

关于本书内容的说明，如果你在哪里看到了与本书雷同的内容，你需要确定一下它的内容是否来自于相应软件的官方网站、man 文档、howto、README，Changelog、INSTALL、LICENSE、*.conf 等，这些是原创，在我看来，什么是原创，只有这些才是原创（我个人的观点，别拿砖头拍我!），我们只不过是对它们的衍生和应用，本书中的内容就是这样，这是我个人的一个学习方法，对于每一个新使用的软件，我都会看它提供的相关文档和其官方网站，配置文件绝对是软件的精华所在，因此在本书中讲述了大量的配置文件，没办法，Linux 下的服务不就是命令加上配置吗?

由于这些官方网站、man 文档、howto、README，Changelog、INSTALL、LICENSE、*.conf 等都是英文的，因此对于我们的认识和阅读是不方便的，事实上我们也正是缺乏这些文档的知识才导致我们一直徘徊在技术的门口，因此本人就是基于这个基础来编写本书，将这些最基础也是最权威的文档通过理解来实现汉语化，以便更多的国人阅读，以个人的感觉，这些东西实际上是我们最需要的，它是认知的第一步，毕竟我们的官方语言是汉语。

书中的内容是我在工作中的一个总结，我没有去刻意改变一个说法，相反只要是官方文档中有的，我就尽可能地采用它们的提法、说法及方法。

与儿子的对话

有一天我儿子问我：“写这套书能挣多少钱？”

这一句话问的不是我，而是问到了人的灵魂，我不知道是否又要归功于教育？有一首歌叫做《我在马路边捡到一分钱》，回忆我小时候，我们学过这首歌，我做到了捡到交公。

我问我儿子：“你学过这首歌吗？”

我儿子说没有学过，他没有学过，但是他也做到了，不但他做到了，而且也帮助他的同学做到了，对于这些我不便多说，报纸有报道。

但是儿子能够问我“写这套书能挣多少钱？”，我确实没有想到。

于是我对他说，写这套书我有三个目的。

第一，对我这么多年做系统管理的一个总结；

第二，帮助那些需要帮助的系统管理和运维人员；

第三，劳动者总是光荣的，出版者给我的报酬是一个物质的体现。

这三个目的是有联系的，没有经验就无法总结，没有总结何谈帮助别人，一个人生活在人世间不只有索取才能感到快乐，相反，给别人带来快乐才是真正的快乐，我希望每个人的奉献要远远大于索取，就是说奉献是我们真正要做的，那么我们的人类才会有真正意义上的和谐，而不是一句口号、一个形式或一个章程！

我儿子又问我：“那干嘛不免费发表在互联网上？”

我对儿子是这样说的：

在这个社会形式下，没有钱是无法生存的。达尔文的进化论说了“适者生存”，达尔文的进化论也说了“要么选择，要么适应”，因此，我们需要适应这个社会。首先我不知道怎么适者生存，但我们要解决自己的温饱问题。

还有一个问题，我发表在网上的文章被别人抄袭了，而且有很多地方都抄错了，最后还有人问我是不是抄袭了别人的，这个很让我郁闷！其实我写在网上的文章就是让别人看的，“天下文章一大抄，就看你会抄不会抄”，这个是小学就明白的道理，技术又何尝不是如此，看看我们所使用的这些技术，有哪个是我们自己的技术，人家外国人什么时候找过我们的麻烦！因此，关于本书的内容，我在“内容声明”中做了特别的说明。总言之，第二个不发表在网上的原因就是本书具有可行性（照着本书，每一个实例都能实现），因此，在转载中的修改对于最终的读者会有障碍。

这是我与我 9 岁儿子的对话。

关 于 读 者

为了使读者快地进入 Nginx 世界，可以从以下四个方面来认识 Nginx。

从功能上要认识到以下五点：

- 提供静态文件和 index 文件，生成自动索引，打开文件描述符缓存；
- 使用缓存加速反向代理，简单的负载平衡和容错；
- 使用缓存机制加速远程 FastCGI 服务器的访问，简单的负载平衡和容错；
- 模块化结构
- 支持 SSL 和 TLS SNI。

对于“邮件代理服务器功能”也可以做适当的了解，毕竟也是 Nginx 的一个功能。

从使用上要认识到以下两点：

于 Nginx 的使用，我们有两点要认识：

- 高并发访问，解决了 10K 的问题；
- 代理，作为代理是它最主要的功能，因此，我们在学习 Nginx 时这是它的主线。

对于 Nginx 的工作机制要认识以下八点：

- 一个 master 进程和几个 workers 进程，workers 进程由非特权用户运行。
- 消息通知方法：kqueue（FreeBSD 4.1+），epoll（Linux 2.6+），rt signals（Linux 2.2.19+），/dev/poll（Solaris 7 11/99+），event ports（Solaris 10），select 和 poll。
- 支持 kqueue 的各种功能，包括 EV_CLEAR，EV_DISABLE（禁用临时事件，NOTE_LOWAT，EV_EOF，number of available data，错误代码。
- 支持 sendfile（FreeBSD 3.1+，Linux 2.2+，Mac OS X 10.5），sendfile64（Linux 2.4.21+），和 sendfilev（Solaris 8 7/01+）。
- File AIO（FreeBSD 4.3+，Linux 2.6.22+）。
- 支持 Accept-filters（FreeBSD 4.1+）和 TCP_DEFER_ACCEPT（Linux 2.4+）。
- 10 000 个非活动 HTTP keep-alive 连接用掉 2.5MB 内存。
- 数据复制操作控制在最低限度。

对于安装平台要认识以下五点：

- FreeBSD 3 — 8 / i386，FreeBSD 5 — 8 / amd64。
- Linux 2.2 — 2.6 / i386，Linux 2.6 / amd64。
- Solaris 9 / i386，sun4u，Solaris 10 / i386，amd64，sun4v。
- MacOS X / ppc，i386。
- Windows XP。Windows Server 2003。

作 者 声 明

本书的内容是我在工作中的一个总结，在生产环境中都使用过，并非纸上谈兵，但是书中的例子，我尽可能地不使用生产环境中的例子，**一是怕对你造成误导，二是不想说什么是权威。**

我在前面说了**文稿内容**的来源，对于文稿的构成，一部分是对员工培训的文稿，一部分是在培训中心的文案，还有一部分是我在学习中的笔记，由这三部分融合而成，而非简单的拼凑。

另外，毕竟我们都是做互联网的，每天面对着无数个页面，我所要说的是，如果读者在阅读本书的过程中发现有和网络上相似的内容，那么确定一下是否是两者（即笔者和您看到大文章的作者）参考了同一个官方的资料，本人绝对没有有意抄袭其他作者的内容，这是第一；第二，如果真的是我写的内容确实是和您的内容有相同之处，那么及时和我联系（绝对是缘分！）；第三，互联网给了我们发展，也给了我们交流，如果您在看本书的过程中发现有个别说法、提法和您的相同，那么请您海涵，往往是一个提法、说法用久了就觉得是自己的说法了（我相信谁都会犯这种错！）；第四，由于本人是做运维（系统管理和网络管理），因此在写作风格上也是按照自己的认知过程所写，既没有受过专业的训练也没有模仿某个作者

或者某个作品的写作风格，如果真的和您的写作风格相同，那么绝对是巧合（这个就不要计较了！）；第五，本套书中引用了互联网的一些内容，由于同一个内容被转来转去，确实很难找到原出处，因此在引用的内容处只指明了来源于互联网。

由于本人才疏学浅，因此，对于本书难免会有疏漏和不足，因此，如果广大读者如果有什么建议和意见可以给本人发邮件：nginx_web_service@126.com。

写 给 读 者

我们告别 20 世纪已经 10 多年了，小的时候老师总对我们说：2000 年后会如何如何，作为 70 后的我，在 2000 年后明白了，生活还得自己去继续，《国际歌》中有一句唱词“从来没有什么救世主”，因此，我很感谢你购买了这套书，希望你能够认真仔细地去阅读。写在书上的叫知识，而被掌握的知识才能转化为财富。我不想大谈什么给人类做多少贡献，只是觉得不要成为社会的拖累和人类的敌人就行了，我们的国家在繁荣富强，在走向文明，但殊不知文明背后就是垃圾，一个塑料袋文明了、方便了，可是看看刮风时漫天的塑料袋，不得不让我想起《水手》中的唱词“那一片被文明糟蹋过的海洋和天地”。再看看我们这个行业，我们每天在玩命地工作，公司或是单位拿着真金白银买着服务器，托管在高档次的 IDC，拼死拼活地去赚钱，3～5 年之后这些服务器将成为垃圾，然后又得去拿着真金白银去买服务器，看看我们服务器的市场，哪个是我们国人生产的，别和我说 XX，那是二道贩子，它只是在挂着自己的羊头在卖别人的狼肉；说完了硬件我们再来看看软件，多亏了我们是在开源中生存，否则就这个软件的费用也受不了，我们还有哪一个是国产的软件，真是让我们无语。20 多年的历程了，拿不出一款与世界抗衡的软件，20 多年的历程了，拿不出与世界抗衡的硬件……

我向来都是把自己看做生活与工作在社会最底层的人，原则就是不成为人类的敌人，不成为国家的拖累，而我们的那些天之骄子哪里去了，能够改变人类、改变生活的人哪里去了！假学说、假学历，假履历，这又让我想起一个冯巩小品中的说辞“这年头有假的谁还用真的”。

伟大领袖毛泽东同志的诗词“天若有情天亦老，人间正道是沧桑”，传统的炼油太不容易，因此很多人就投资炼地沟油去了，十种地沟油八种合格，想的都让人跳楼，是化验的不准，还是这种地沟油就是合格，如果要是合格那么就是可以使用了，这再好不过了，废物能够再利用，那是人类的功臣才对，还可以为人类做很大的贡献；如果是化验不准，那我们的科研人员都干什么去了，是为了节约国家运营成本裁掉了？

好了，写给读者的就这些，我不再多说了，就让我们以“天若有情天亦老，人间正道是沧桑”共勉！

陶利军

目　　录

第 1 部分　Nginx 服务器

第 2 部分 Nginx 服务器的功能模块

第 3 部分　Nginx 与缓存

第 1 部分

Nginx 服务器

作为本书的第一部分，我们首先要了解 Nginx 服务器的基本功能、管理方式、如何处理一个请求，以及一些用于更高管理 Nginx 服务器的工具，还有要为作为网站第一道门的 Nginx 服务器实现高可用。

Nginx 是一个自由的、开源的、高性能的 HTTP 服务器和反向代理，同时也是一个 IMAP/POP3 代理服务器。它是由 Igor Sysoev 于 2002 年开发，并且在 2004 年发布了第一个版本。在互联网上使用 Nginx 的主机近乎 6.55%。

Nginx 之所以能够脱颖而出、闻名世界，是因为它的高性能、高稳定性、丰富的功能设置、简单的配置和低的资源消耗。

Nginx 解决了服务器的 C10K 问题。它的设计不像传统的服务器那样使用线程处理请求，而是使用了一个更加高级的机制——事件驱动机制，是一种异步事件驱动结构。

即使你不希望处理成千上万的并发请求，同样能够从 Nginx 的高性能和低消耗内存（占用内存小）的结构中获益。Nginx 的使用规模很全面：从很小的 VPS 到服务器集群都可以使用。

Nginx 强有力地用在了一些高知名度的站点，例如 WordPress、Hulu、Github、Ohloh、SourceForge 和 TorrentReactor。

第 1 章　Nginx 的功能

本章我们来认识 Nginx 服务器的基本功能和扩展功能，以及 Nginx 核心模块的相关指令和变量。

1.1　功能描述

Nginx 的功能包括基本 HTTP 功能和扩展功能。和 Apache 服务器一样，Nginx 服务器为了提供更多的功能并且能够有效地扩展这些功能，使用了模块化的方式来扩展其功能。每一个模块都提供了一个功能，通过编译这些功能模块来实现功能的扩展。

1.1.1　基本 HTTP 功能

- 提供静态文件和 index 文件，生成自动索引，打开文件描述符缓存；
- 使用缓存加速反向代理，简单的负载平衡和容错；
- 使用缓存机制加速远程 FastCGI 服务器的访问，简单的负载平衡和容错；
- 模块化的结构，过滤器包括 gzip、字节 range、chunk 响应、XSLT、SSI 和图像大小调整（确切地说是将大图转换为小图）过滤，被传递到后台服务器（FastCGI 或者是代理服务器）多个 SSI 指令在单个页面的并行处理；
- 支持 SSL 和 TLS SNI。

1.1.2　其他 HTTP 功能

- 基于名称和基于 IP 的虚拟服务器；
- 支持 Keep-alive 和 管道连接；
- 灵活的配置；
- 重新配置和在线升级而不用中断对客户访问的处理；
- 访问日志的格式，缓存日志写入和快速日志轮循；
- 3xx-5xx 错误代码重定向；
- 重写模块；
- 基于客户端 IP 地址和 HTTP 基本认证的访问控制；
- 基于 HTTP 协议的 PUT，DELETE，MKCOL，COPY 和 MOVE 方法；
- FLV 流文件；
- 速度限制；
- 限制同时连接数或者是来自同一 IP 地址的请求；
- 嵌入式 Perl 语言解析。

1.1.3 邮件代理服务器功能

- 使用外面的 HTTP 认证服务器提供认证，然后重定向到后台内部的 IMAP/POP3 服务器；
- 使用外面的 HTTP 认证服务器提供认证，然后重定向到后台内部的 SMTP 服务器；
- 认证方法；
- POP3：USER/PASS，APOP，AUTH LOGIN/PLAIN/CRAM-MD5；
- IMAP：LOGIN，AUTH LOGIN/PLAIN/CRAM-MD5；
- SMTP：AUTH LOGIN/PLAIN/CRAM-MD5；
- SSL 支持；
- 支持 STARTTLS 和 STLS。

1.1.4 架构和可扩展性

- 一个 master 进程和几个 workers 进程，workers 进程由非特权用户运行；
- 消息通知方法：kqueue（FreeBSD 4.1+），epoll（Linux 2.6+），rt signals（Linux 2.2.19+），/dev/poll（Solaris 7 11/99+），event ports（Solaris 10），select, and poll；
- 支持 kqueue 的各种功能，包括 EV_CLEAR，EV_DISABLE（禁用临时事件），NOTE_LOWAT，EV_EOF，number of available data，错误代码；
- 支持 sendfile（FreeBSD 3.1+，Linux 2.2+，Mac OS X 10.5），sendfile64（Linux 2.4.21+），和 sendfilev（Solaris 8 7/01+）；
- 文件 AIO（FreeBSD 4.3+，Linux 2.6.22+）支持；
- 支持 Accept-filters（FreeBSD 4.1+）和 TCP_DEFER_ACCEPT（Linux 2.4+）；
- 10 000 个非活动 HTTP keep-alive 连接用掉 2.5MB 内存；
- 数据复制操作控制在最低限度。

因此，总结如下：

- 非阻塞；
- 事件驱动；
- 单线程；
- 一个 master 和几个 worker；
- 高效的资源使用；
- 高度的模块化。

1.1.5 被测试的系统和平台

- FreeBSD 3～8 / i386，FreeBSD 5～8 / amd64；
- Linux 2.2～2.6 / i386，Linux 2.6 / amd64；
- Solaris 9 / i386，sun4u，Solaris 10 / i386，amd64，sun4v；
- MacOS X / ppc，i386；
- Windows XP，Windows Server 2003。

1.2 服务器的类型

在一个大型的网站构建中包括了很多服务器类型，它们在整个访问中提供了不同的功能，发挥着不同的作。下面我们来看一下服务器的类型。

1. Web 服务器

Web 服务器用于提供 HTTP（包括 HTTPS）的访问，例如 Nginx、Apache、IIS 等，虽然 Tomcat 也能够做到，但这并不是它的主要功能，而且其性能也不如专门的 Web 服务器。

2. 应用程序服务器

应用程序服务器能够用于应用程序的运行，包括的工作有：客户会话管理、业务逻辑管理、数据操作等。

3. 代理服务器

代理服务器通常是客户端访问的一种行为。它虽然不属于网站部署中的环境，但在整体的客户端访问中，它却是一个重要环节的服务器。

4. 反向代理

与代理服务器相对，还有一个反向代理服务器，其功能就是 Web 服务器的功能。但是从它这里拿到的网页不是最原始产生页面的“产生地”，而是由它从页面的“产生地”取回页面后的一个缓存。代理服务器中缓存的内容通常是由某些用户访问某个页面而产生访问请求后，在客户端代理服务器上留下的缓存；而反向代理服务器上缓存的页面，不是由于某些用户访问某个页面后留下的缓存，却是根据网站运维的策略定期、定时地生成一些后台服务器的页面缓存。

代理服务器和反向代理服务器的区别并不在于以上的这种却别，两者真正的区别在于代理服务器是工作在客户端的，而反向代理服务器是工作在服务器端的。在反向代理服务器中，Nginx 是最优越的，这也是我们使用其功能最多的一个功能。

5. 后台服务器

后台服务器只是一个说法而已，这是根据它的工作特点来说的，换句话说就是没有直接提供给客户访问。例如众多的 FastCGI 服务器，它们都工作在后台，HTTP 协议却无法访问到它们，另一种情况，如果我们从前是通过使用 Apache 作为 Web 服务器提供 HTTP 访问的，现在被 Nginx 反向代理了，就是说由 Nginx 直接面对客户访问，而将请求再转到 Apache 服务器，那么这里的 Apache 服务器就已经成为后台服务器了。

6. CDN 缓存服务器

正如其名字，它就是缓存服务器的角色，而且是反向代理的应用，在网站部署中，它算是一种部署策略，即对于远距离访问的解决方案，为了解决时间产生距离、时间缩短距离而产生的，它就是反向代理的另一种应用。

Nginx 服务器可以胜任其中的每一种服务器。

1.3 认识 Nginx 服务器的基本模块

Nginx 是基于模块化的构建方式。

1. 从功能上划分

- Nginx 核心模块：包括 Nginx 的内核模块和事件驱动模块；
- Nginx 邮件模块：包括 Mail 的内核模块和相关的认证、代理，以及提供 POP3、IMAP 和 SMTP 的 SSL 模块；
- HTTP 服务模块：这类模块包括三类模块，即 HTTP 的内核模块和标准模块以及可选的 HTTP 模块。

2. 从发布模块的方式来划分

- 官方模块；
- 第三方模块。

3. 从模块的可选项来划分

- 必选模块；
- 可选模块。

最后一种划分方法看似简单，但实际上却经常被忽略，在安装部署 Nginx 服务器时，一定要遵循：需要某一个模块则安装，不需要则不要安装，每一个被安装的模块都会影响 Nginx 的性能和占用系统资源。

本章将介绍 Nginx 的两个核心模块和 HTTP 核心模块，通过这三个模块的学习来认识概念上的 Nginx。

有关 Nginx 的功能模块将在第 2 部分进行详细介绍和应用，关于 Nginx 与其他应用程序服务器将会放在本套书的卷 2 中。卷 2 中将会根据现有的应用程序与 Nginx 服务器的结合实现动静分离。

下面来认识一下 Nginx 服务器的基本模块。前两个模块是 Nginx 的内核模块和事件驱动模块，即 CoreModule 和 EventsModule；第三个模块是 Http 内核模块，即 HttpCoreModule，它是 Nginx 服务器的核心模块。

相对于 HttpCoreModule 模块来说，CoreModule 和 EventsModule 模块的配置会少些，但是它们的配置将会影响系统的性能，而非功能上的差异。因此，我们将通过这些模块的指令来认识它们的作用。这些模块也是 Nginx 必需的模块。

1.3.1 Nginx 的内核模块

Nginx 的内核模块用于控制 Nginx 服务器的基本功能。

1. 配置示例

```
#user  nobody;
worker_processes  1;

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;
#pidlogs/nginx.pid;
```

2. 指令

下面这些指令中，大部分指令必须放置在配置文件的根部，即配置文件的开始部分，而且只能使用一次。然而，有些却在多种情况下有效，如果有这种情况，那么在列表中会有说明，说明该指令会在哪些环境中可用，在配置文件的根部却只能使用一次。

指令名称：daemon

语法：daemon on | off

默认值：on

功能：在生产环境下不要使用 daemon 和 master_process 指令，这些指令只用于开发环境。虽然在生产环境中可以使用 daemon off，但对性能的提升没有任何帮助；在生产环境中永远不要使用 master_process off。另外，正如指令的名称，需要注意的一点是，如果使用“daemon off”，那么 Nginx 将会运行在前台，而不会运行在后台的守护进程，例如：

```
[root@mail conf]#/usr/local/nginx-0.8.54/sbin/nginx
```

Nginx 进程将会运行在前台。

指令名称：env

语法：env VAR|VAR=VALUE

功能：该指令用于对环境变量重新定义。

例如：

```
env  MALLOC_OPTIONS;
env  PERL5LIB=/data/site/modules;
env  OPENSSL_ALLOW_PROXY_CERTS=1;
```

指令名称：debug_points

语法：debug_points [stop | abort]

默认值：none

功能：激化所有设置的调试点。

指令名称：error_log

语法：error_log file [debug | info | notice | warn | error | crit]

默认值：${prefix}/logs/error.log

使用环境：http、server 和 location

功能：该指令用于 Nginx 服务器（包括 FastCGI）指定记录错误日志的文件和记录错误的级别。日志的级别有 debug、info、notice、warn、error 和 crit（详细程度由高到低：debug 提供了全部日志记录，而 crit 仅报告了关键错误）。

需要注意的是，不要认为设置为 error_log off 则能够关闭日志记录功能，相反这样会将日志文件写入到一个文件名为 off 的文件中。如果想关闭错误日志记录功能，则可以使用以下配置：

```
error_log /dev/null crit;
```

指令名称：include

语法：include file | *

默认值：none

功能：该指令用于载入配置文件。需要注意的是，如果没有指定绝对路径，那么文件的路径将会和配置文件的目录相关，换言之，Nginx 会认为其与配置文件在同一目录下。例如，“include www.xx.cn.conf”，那么 Nginx 则会认为实际的文件包含在以下目录：/usr/local/nginx/conf/sites/example.conf。

需要说明的一点是：在 0.6.7 版本以后指定的文件相对路径根据 nginx.conf 所在的目录来决定，而不是 prefix 目录的路径；之前的版本则是根据 --prefix=PATH 来决定。

另外，从 0.4.4 版本以后，include 指令已经能够支持文件通配符，例如：

```
include vhosts/*.conf;
```

指令名称：lock_file

语法：lock_file file

默认值：编译时指定。

功能：Nginx 使用了连接互斥锁进行顺序的 accept () 系统调用，如果 Nginx 使用 gcc、Intel C++或者是 SunPro C++在 i386、amd64、sparc64 和 ppc64 编译创建，那么 Nginx 服务器将自动采用异步互斥进行访问控制，而在其他情况下锁文件会被使用，默认是不使用，除非在编译时开启了该功能。

例如：

```
lock_file  /var/log/lock_file;
```

指令名称：master_process

语法：master_process on | off

默认值：on

功能：如果设置为 on，那么 Nginx 将会开启多个进程，包括一个主进程（就是 master 进程）和多个 worker 进程；如果设置为 off 即禁用，那么 Nginx 将会以独一无二的进程，即 master 进程来运行。该指令仅被用于测试，作为一个 master 进程，这样客户端就不能连接到你的服务器。因此在生产环境中不要设置该选项。

指令名称：pid

语法：pid 文件

默认值: 编译时指定。

功能：该指令用于设置 Nginx 的 pid 文件。通过使用 kill 命令向该文件发送相应的信号来结束 Nginx 运行、重新载入配置等操作，例如：

```
kill -HUP'cat /var/log/nginx.pid'
```

例如：

```
pid /var/log/nginx.pid;
```

指令名称：ssl_engine

语法：ssl_engine engine

默认值：依赖于系统。

功能：该指令用于设置想要使用的 OpenSSL 引擎，可以通过命令“openssl engine -t”来查找可用的引擎。例如：

```
$ openssl engine -t
 (cryptodev) BSD cryptodev engine
  [ available ]
 (dynamic) Dynamic engine loading support
  [ unavailable ]
```

指令名称：timer_resolution

语法：timer_resolution t

默认值：none

功能：该指令用于缩短 gettimeofday（）系统调用的时间，默认情况下 gettimeofday（）在 kevent（）、epoll、/dev/poll、select（）及 poll（）调用完成之后调用。如果在具体的使用中需要一个比较准确的时间来记录$upstream_response_time 或者$msec 变量，那么将会使用到该指令。

例如：

```
timer_resolution  100ms;
```

指令名称：user

语法：user user [group]

默认值：nobody nobody

功能：如果 master 进程是以 root 用户来运行的，那么 Nginx 将会使用 setuid（）/setgid（）来实现 USER/GROUP 的接替工作，如果没有指定 GROUP，那么 Nginx 将会使用同 USER 一样的组名称。默认情况下，使用 nobody 用户名称 和 nobody 或者是 nogroup 组名称，当然也可以在配置./configure 脚本时指定--user=USER 和 --group=GROUP 来实现。

例如：

```
user www users;
```

指令名称：worker_cpu_affinity

语法：worker_cpu_affinity cpumask [cpumask...]

默认值：none

功能：该指令只能用于 Linux，设置 worker 进程与 CPU 的亲和力。该指令允许通过调用 sched_setaffinity（）将 worker 进程绑定到一个 CPU 上。

例如：

```
worker processes 4;
worker_cpu_affinity 0001 0010 0100 1000;
```

这种设置将每一个进程绑定到一个 CPU 上。

```
worker_processes 2;
worker_cpu_affinity 0101 1010;
```

这种设置将第一个 worker 绑定到 CPU0/CPU2；将第二个 worker 绑定到 CPU1/CPU3。

这对于超线程（HTT）CPU 合适。

指令名称：worker_priority

语法：worker_priority [-] number

默认值：on

功能：该指令用于指定 worker 进程的优先级，从-20（最高级）到 19（最低级），默认值为 0。

注意 kernel 进程运行在-5 优先级，因此不建议设置-5 或较小。

指令名称：worker_processes

语法：worker_processes number

默认值：1

功能：如果 Nginx 提供 SSL 或者是 gzip，即对 CPU 的使用率较高，并且系统中有两个以上的 CPU 或者内核，那么可以设置 worker_processes 的值为 CPU 的数量或者是内核的数量。如果提供了大量的静态文件，并且总的数量超过了有效内存的大小，那么可以增加该指令的值，以便充分利用磁盘的带宽。另外，如果想要将 worker 进程绑定到某个 CPU 或者内核上，则可以使用 worker_cpu_affinity 指令。

由于以下的原因，Nginx 可能要使用多个 worker 进程：

- 使用 SMP；
- 在 worker 进程阻塞磁盘 I/O 时，设置多个 worker 进程可以减少延时，具体来说就是如果一个 worker 进程由于慢的 I/O 操作被阻塞，那么进入的请求会被转交给其他的 worker 进程；
- 当使用了 select（）/poll（）限制了每一个进程的连接数时。

例如：

```
worker_processes 4;
```

指令名称：worker_rlimit_core

语法：worker_rlimit_core size

功能：用于定义每个进程核心文件的最大值，主要用于 debug。

指令名称 worker_rlimit_nofile

语法：worker_rlimit_nofile limit

默认值：No value specified, so OS default

功能：该指令用于定义一个 worker 进程可以同时处理的文件数量。

指令名称：worker_rlimit_sigpending

语法：worker_rlimit_sigpending limit

功能：定义每个用户（调用进程的用户 ID）能够被排入队列的信号（signals）数量。如果队列（queue）满了，但由于这个限制，信号（signals）会被忽略。

指令名称：working_directory

语法：working_directory 路径

默认值：依赖于--prefix 的值

功能：该指令用于设定 worker 进程工作的目录，仅用于定义核心（core）文件的位置。对于该目录，worker 进程用户（user 指令指定的用户）必须有写的权限，用于能够写

入核心（core 文件）。

3. 变量

Nginx 的内核模块提供了以下两个变量。

变量名称：$pid

功能：该变量表示当前 Nginx 服务器的进程 ID 号。

变量名称：$realpath_root

功能：没有找相应的文档资料。

从这些指令中可以看出，有些指令只是在开发阶段使用，而在将 Nginx 服务器部署到生产环境时就不再使用它们了。

4. 使用配置实例

```
user  web;
worker processes  4;

error log  /var/log/nginx/error.log;
includevhosts/*.conf;
pidlogs/nginx.pid;
```

1.3.2 Nginx 的事件模块

事件模块（EventsModule）用于控制 Nginx 如何处理连接。该模块的指令即指令的一些参数会对应用程序的性能产生重要的影响。因此在设置时要慎重。

1. 配置示例

```
events {
worker_connections  1024;
}
```

2. 指令

Nginx 的事件模块提供了以下指令，所有这些指令只能在 events 区段设置。

指令名称：accept_mutex_delay

语法：accept_mutex_delay Nms;

默认值：500ms

功能：如果一个工作进程（worker process）没有互斥锁，那么它将至少在这个设定值的时间之后才会被回收。

指令名称：debug_connection

语法：debug_connection [ip | CIDR]

默认值：none

功能：该指令用于指定只记录由某个 IP 地址或者某个网段的客户端产生的 debug 信息，可以指定多个参数。从 0.3.54 版本之后，这个参数可以支持 CIDR 地址格式。

例如：

```
error_log /var/log/nginx/errors;
events {
  debug_connection  192.168.1.1;
}
```

指令名称：worker_connections

语法：worker_connections number

默认值：1024

功能：该指令用于设置每个 worker 进程所能处理的连接数。

通过 worker_connections 和 worker_proceses 指令能够计算出最大客户端连接数：

```
max_clients = worker_processes * worker_connections
```

在反向代理的环境中，最大客户端连接数变为：

```
max_clients = worker_processes * worker_connections/4
```

原因在于，默认情况下一个浏览器会对服务器打开两个连接，Nginx 使用来自于同一个池中的 FDS（文件描述符）来连接上游服务器。

指令名称：connections

功能：该指令已被 worker_connections 取代，不提倡继续使用。

指令名称：use

语法：use type

默认值：在编译时指定。

功能：如果在执行./configure 的时候指定了不只一个事件模型，那么在使用 Nginx 时可以通过该指令告诉 Nginx 想使用的事件驱动模型。默认情况下，Nginx 在编译时会检测系统，并且 Nginx 会根据所在操作系统选择一个最合适的事件驱动类型。可选择的值：/dev/poll、epoll、eventport、kqueue、rtsig 或 select。下面是事件驱动类型。

- /dev/poll：一种用于 Solaris 7 11/99+，HP/UX 11.22+，IRIX 6.5.15+，Tru64 UNIX 5.1A+系统的高效模型；
- epoll：一种基于 Linux 2.6+ 操作系统下有效的模型；
- eventport：用于 Solaris 10 的一种高效的模型，但是需要安全补丁；
- kqueue：一种在 FreeBSD 4.1+，OpenBSD 2.9+，NetBSD 2.0，MacOS X 操作系统下性能高效的模型；
- rtsig：实时信号，对于 Linux 2.2.19 有效，但对于高流量的情况则不适应，默认情况下，系统仅允许 1 024 个队列信号；
- select：默认的标准模块，如果 OS 不支持更有效的模型，那么它将被使用（这种模型也是 Windows 下仅有的一种模型）；
- poll：在自动选择上其优先于 select，但在所有的系统上都无效。

指令名称：multi_accept

语法：multi_accept [on | off]

默认值：off

功能：定义 Nginx 是否立刻接收从所有监听队列进入的连接。也就是说 multi_accept 会在 Nginx 接到一个新连接后立即发出通知后调用 accept（）来接收尽量多的连接。

指令名称：accept_mutex

语法：accept_mutex [on | off]

默认值：on

功能：Nginx 使用连接互斥锁（mutex）进行顺序的 accept（）系统调用。

指令名称：accept_mutex_delay

语法：accept_mutex_delay Nms;

默认值：500

功能：该指令用于定义一个 worker 进程在尝试再次获取资源之前应该等待的时间。如果指令 accept_mutex 设置为 off，那么该值（指的是指令 accept_mutex_delay 的值）不能被使用。单位为毫秒（milliseconds）。

3. 使用实例

```
events {
    worker connections 1024;
    use epoll;
    worker_connections 32768
}
```

1.3.3 Nginx 的 HTTP 内核模块

认识 Nginx 的 HTTP 功能首先要从它的 HTTP 核心模块（HttpCoreModule）开始，在安装 Nginx 的过程中对于一个提供 HTTP 访问的 Nginx 服务器，该模块是不能够被禁用的。

1. 配置结构

```
http {

...

server {
listen   80;
server_name  www.yy.cn;

...

location / {
root   html;
index  index.html index.htm;
}
```

```
...

}


server {

...

}

# HTTPS server

server {
listen   443;
server_name  www.xx.cn;

ssl  on;
ssl_certificate  cert.pem;
ssl_certificate_key  cert.key;

ssl_session_timeout  5m;

ssl_protocols  SSLv2 SSLv3 TLSv1;
ssl_ciphers  ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
ssl_prefer_server_ciphers   on;

location / {
root   html;
index  index.html index.htm;
}
}

}
```

2. 指令

指令名称：aio

语法：aio [on|off|sendfile]

默认值：off

使用环境：http，server，location

功能：该指令在 Linux 内核 2.6.1922 以上版本可以使用，对于 Linux 来说还需要配合使用 directio 指令。另外，如果使用了该指令，那么将会自动禁用 sendfile 的支持。例如：

```
location /video {
aio on;
directio 512;
output buffers 1 128k;
}
```

在 FreeBSD 5.2.1 之前的版本和 Nginx 0.8.12 版本，必须禁用 sendfile 的支持：

```
location /video {
aio on;
sendfile off;
output_buffers 1 128k;
}
```

从 FreeBSD 5.2.1 起和 Nginx0.8.12 版本可以一起使用 aio 和 sendfile，例如：

```
location /video {
aio sendfile;
sendfile on;
tcp_nopush on;
}
```

指令名称：alias

语法：alias file-path|directory-path

默认值：no

使用环境：location

功能：该指令用于指定一个路径，但是它不同于 root 指令，我们将通过例子来说明它们的区别。

例如：以下是在 Nginx 中添加的配置：

```
location  /i/ {
  alias  /spool/w3/images/;
}

location  /m/ {
  root  /spool/w3/images/;
}
```

目录结构：

```
/spool/w3/images/
'-- 8.jpg

0 directory, 1 files
```

访问测试：

访问 http://www.xx.com/i/8.jpg，可以查看 Nginx 的访问日志：

```
   192.168.100.253 - - [20/Oct/2011:10:14:21 +0800] "GET /i/8.jpg HTTP/1.1"
200 11652 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

可以看到访问是成功的！

继续访问 http://www.xx.com/i/8.jpg，来查看 Nginx 的访问日志：

```
192.168.100.253 - - [20/Oct/2011:10:23:20 +0800] "GET /m/8.jpg HTTP/1.1"
404 169 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
```

这次访问以 404 失败告终，也就是这次没有找到要访问的文件。

下面我们更改一下目录结构：

```
/spool/w3/images/
|-- 8.jpg
'-- m
'-- 8.jpg

1 directory, 2 files
```

再次访问 http://www.xx.com/i/8.jpg，来查看 Nginx 的访问日志：

```
192.168.100.253- - [20/Oct/2011:10:25:22 +0800] "GET /m/8.jpg HTTP/1.1" 200
11652 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

这次访问成功！

分析如下：

如果客户端请求"/i/8.jpg"，那么根据这个配置将会由"/spool/w3/images/8.jpg"文件来提供客户端的访问。而如果客户端请求"/m/8.jpg"，那么根据这个配置将会由"/spool/w3/images/m/8.jpg"文件来提供客户端的访问，而不是"/spool/w3/images/8.jpg"。

因此，我们的结论是，root 指令的文档路径，将会使得访问者请求的 URL（例如这里的 m/）也就是 location 中的"/m/"，将会附加到 root 指令指定的路径之后，然后再去查找文件。简言之，就是 location 中的"/m/"要附加在"/spool/w3/images/"之后。而 alias 则不是，客户端请求中的 URI（例如这里的 m/）将会直接映射到 alias 指令指定的"/spool/w3/images/"目录中。

另外，也可以在 location 中使用正则表达式，例如：

```
location ~ ^/download/(.*)$ {
  alias /home/website/files/$1;
}
```

在这种配置下，如果请求"/download/book.pdf"，那么返回的文件将会是"/home/website/files/book.pdf"。

> 注意，只有请求部分 URI 位置的内容追加别名定义的路径之后。另外还需要注意的是，也可以在别名目录字段中使用变量。

指令名称：chunked_transfer_encoding

语法：chunked_transfer_encoding on|off

默认值：on

使用环境：http，server，location

功能：该指令设置了是否在响应中使用 chunk 编码，从 0.7.66 以后的 Nginx 版本中提供了

这个指令，另外要想使用 chunk 编码需要在 HTTP1.1 协议下访问才有效。

指令名称：client_body_in_file_only

语法：client_body_in_file_only on|off

默认值：off

使用环境：http，server，location

功能：该指令总是强制 Nginx 将客户端请求体存储到一个临时的磁盘文件，即使请求体的实际大小为 0。然而需要注意的是，在启用该指令之后，该文件在请求完成之后并不会被移除。该指令可以用于调试和嵌入式 Perl 模块$r->request_body_file 方法的使用。

指令名称：client_body_in_single_buffer

语法：client_body_in_single_buffer

默认值：off

使用环境：http，server，location

功能：该指令在 0.7.58 以上的 Nginx 版本中提供，用于指定是否将整个客户端请求体保存在单个请求缓存中。为了减少复制操作，当使用变量$request_body 的时候，推荐使用该指令。

> 注意：当请求体不能被单个缓存（参考 client_body_buffer_size）容纳下的时候，那么请求体仍将会保存到磁盘上。

指令名称：client_body_buffer_size

语法：client_body_buffer_size the_size

默认值：8k/16k

使用环境：http，server，location

功能：该指令指定了客户端请求体缓存的大小。如果请求体大于该缓存大小，那么整个请求体或者请求体的某些部分将会被写入临时文件。

默认值等于两个页面的大小，页面的大小依赖于所在的操作系统平台，可能是 8k 或者是 16k。

当请求头中 Content-Length 的值小于指定缓存的大小时，那么 Nginx 将会使用较小的一个缓存，因此 Nginx 也并非总是为每一个请求分配指定大小的缓存。

指令名称：client_body_temp_path

语法：client_body_temp_path dir-path [level1 [level2 [level3]

默认值：client_body_temp

使用环境：http，server，location

功能：该指令用于指定一个存储临时文件的目录，在这个目录中将会存储客户端请求体。按照指定的子目录等级，可能会有三级目录。例如：

```
client_body_temp_path  /spool/nginx/client_temp 1 2;
```

那么该目录的存储架构将会是：

```
/spool/nginx/client_temp/7/45/00000123457
```

指令名称：client_body_timeout

语法：client_body_timeout time

默认值：60

使用环境：http，server，location

功能：该指令用于设置读取超时，就是从客户端传入的请求体读取超时。这个超时仅指一个请求的请求体还没有进入请求体的读取步骤时的超时，如果在这个设定的时间内客户端没有发送任何数据，那么 Nginx 将会返回"Request time out" （408）。

指令名称：client_header_buffer_size

语法：client_header_buffer_size size

默认值：1k

使用环境：http，server

功能：该指令用于设置缓存头的大小，这个缓存用于存放从客户端发送到 Nginx 服务器的请求头。对于绝大多数的请求，1k 的缓冲区大小是完全足够的。然而如果有大的 cookie 在请求头中或者是请求头是来自于一个 wap 客户端的请求头，那么 1k 的大小可能不能够容纳下这种请求头的，因此 Nginx 将会为其分配一个大一点的缓存，这个值可以使用 large_client_header_buffers 指令设置。

指令名称：client_header_timeout

语法：client_header_timeout time

默认值：60

使用环境：http，server

功能：该指令用于设置读取客户端请求标题的超时。这个超时仅指一个请求的请求头还没有进入请求头的读取步骤时的超时，如果在这个设定的时间内客户端没有发送任何数据，那么 Nginx 将会返回"Request time out"（408）。

指令名称：client_max_body_size

语法：client_max_body_size size

默认值：client_max_body_size 1m

使用环境：http，server，location

功能：该指令用于设定接受客户端请求体的最大值，在客户端请求头中通过 Content-Length 表明。如果客户端请求体大于指定的值，那么客户端将会收到一个"Request Entity Too Large" （413）的错误。

指令名称：default_type

语法：default_type MIME-type

默认值：default_type text/plain

使用环境：http，server，location

功能：该指令用于指定一个默认的 MIME 类型，对于没有标准 MIME 匹配的文件类型将会使用该类型。

指令名称：directio

语法：directio [size|off]

默认值：directio off

使用环境：http，server，location

功能：该指令用于启用 O_DIRECT（FreeBSD，Linux），F_NOCACHE（Mac OS X）标志或者 directio（）函数（Solaris），对于大于指定大小的文件，那么将启用这种方式读取该文件。如果启用该指令，则将禁用 sendfile。例如：

```
directio  4m;
```

指令名称：error_page

语法：error_page code [code...] [= | =answer-code] uri | @named_location

默认值：no

使用环境：http，server，location，If in location

功能：该指令用于指定一个 URI，在访问出错的时候会显示该网页。

例如:

```
error page   404  /404.html;
error page   502 503 504  /50x.html;
error page   403  http://example.com/forbidden.html;
error_page   404  = @fetch;
```

而且还可以通过该指令将一个状态代码改变为另一个状态代码，例如：

```
error page 404 =200 /empty.gif;
error_page 404 =403 /forbidden.gif;
```

另外，我们还可以通过等号‘=’来使用自己设计的错误处理程序来决定指定错误代码的返回状态，例如：

```
error_page   404 = /404.php;
```

注意，这个“=”后没有状态代码。

如果在重定向中不需要改变 URI，那么可以通过以下方法将错误处理传递到一个命名 location 中：

```
location / (
error_page 404 @fallback;
)

location @fallback (
proxy pass http://backend;
)
```

指令名称：if_modified_since

默认值：if_modified_since exact

语法：if_modified_since [off|exact|before]

使用环境：http，server，location

功能：该指令于 0.7.24 以上的 Nginx 中提供，它的功能在于定义了如何比较修改时间和请求头中“If-Modified-Since”的时间。

- Off：不检测“If-Modified-Since”请求头（0.7.34）；
- Exact：精确匹配；
- Before：文件修改时间应该小于请求头“If-Modified-Since”的时间。

指令名称：internal

语法：internal

默认值：no

使用环境：location

功能：该指令起到一个指示的作用，如果在一个 location 中使用了该指令，那么表示这个 location 只能由“内部”请求访问。对于外部的访问（例如 我们通过浏览器访问）则返回错误"Not found"（404）。内部请求有以下几种方法。

- 使用 error_page 重定向指令；
- 由“ngx_http_ssi_module”模块的 include virtual 指令创建的子请求；
- 由“ngx_http_rewrite_module”模块的 rewrite 指令改变的请求方向。

例如：

```
error_page 404 /404.html;
location  /404.html {
  internal;
}
```

在这个例子中，阻止了客户端直接获取错误页，而是重定向到另一个进一步处理的 Location 中。

指令名称：keepalive_disable

语法：keepalive_disable [msie6 | safari | none]...

默认值：msie6 safari

使用环境：http，server，location

功能：该指令用于禁用某些用户带来的 keepalive 功能，在 0.9.0 以上的版本中提供了该指令。默认情况下，对于 MSIE（低于 6.0 service pack 2）和 Safari 的 keepalive 功能被禁用。

指令名称：keepalive_timeout

语法：keepalive_timeout [time] [time]

默认值：keepalive_timeout 75

使用环境：http，server，location

功能：该指令有两个参数，第一个参数用于设定客户端的 keep-alive 连接超时，在这个时间过后，服务器将会关闭连接。第二个选项是一个可选项，它的值决定了响应头 Keep-Alive：timeout="time"的值，这个头能够告诉一些浏览器关闭连接，这样服务器就不用再次关闭连接了。如果没有设定这个参数，那么 Nginx 将不会发送 Keep-Alive 头。两个参数的值可以不相同。

下面是一些浏览器如何处理 Keep-Alive 头的方法。

- MSIE 和 Opera 忽略“Keep-Alive: timeout=<N>”头。
- MSIE 保持连接的时间大约是 60～65 秒，然后发送一个 TCP RST。
- Opera 将会保持一个较长时间的连接。
- Mozilla 将会在该指令原有设定的基础（例如 N）之上再增加 1～10 秒。
- Konqueror 保持连接 N 秒。

指令名称：keepalive_requests

语法：keepalive_requests n

默认值：keepalive_requests 100

使用环境：http，server，location

功能：该指令用于设置 Nginx 服务器能够保持活跃的连接数。

指令名称：large_client_header_buffers

语法：large_client_header_buffers number size

默认值：large_client_header_buffers 4 4k/8k

使用环境：http，server

功能：该指令用于指定客户端一些比较大的请求头使用的缓冲区数量和大小。请求头行不能够大于一个缓存的大小，如果客户端发送了一个较大的头，那么 Nginx 将会返回一个"Request URI too large"（414）的错误信息。最长的请求头行也必须不能超过一个缓存的大小，否则将会返回"Bad request"（400）错误信息。一个缓存的默认大小等于一个内存页面的大小，这依赖于不同的平台，有的是 4k，而有的是 8k，如果在请求连接的结尾状态转换为 keepalive，那么所占用的这些缓存将会被释放。

指令名称：limit_except

语法：limit_except methods {...}

默认值：no

使用环境：location

功能：该指令用于限制访问 location 的 HTTP 方法。

例如：

```
limit except  GET {
  allow  192.168.1.0/32;
  deny   all;
}
```

指令名称：limit_rate

语法：limit_rate speed

默认值：no

使用环境：http，server，location，if in location

功能：参考“第 11 章　限制流量”。

指令名称：limit_rate_after

语法：limit_rate_after time

默认值：limit_rate_after 1m

使用环境：http，server，location，if in location

功能：参考“第 11 章　限制流量”。

指令名称：lingering_close

语法：lingering_close on|off|always

默认值：lingering_close on

使用环境：http，server，location

功能：用于在 Socket 上启用 SO_LINGER。

指令名称：lingering_time

语法：lingering_time time

默认值：lingering_close 30s

使用环境：http，server，location

功能：该指令会将一个请求体（request body）运用到客户端请求。上传的数据一旦超过 max_client_body_size 指定的值，Nginx 就立即发送"413 Request entity too large"的 HTTP 响应错误。然而，大多数浏览器不管那个通知，继续上传数据。该指令定义了 Nginx 应该等待的时间总数，包括从发送这个错误响应之后到关闭该连接之前的时间。

指令名称：lingering_timeout

语法：lingering_timeout time

默认值：lingering_timeout 5s

使用环境：http，server，location

功能：该指令定义了 Nginx 在客户端关闭之前，两个读操作之间的等待时间。

指令名称：listen

语法：listen address:port [default（0.8.21 起不再使用） | default_server | [backlog=num | rcvbuf=size | sndbuf=size | accept_filter=filter | deferred | bind | ipv6only=[on|off] | ssl]]

默认值：80

使用环境：server

功能：该指令用指定在 Server {...}区段设置接收请求的 IP 地址和端口。可以仅指定一个 IP 地址、一个端口或者是一个服务器名字作为地址，例如：

```
listen 127.0.0.1:8000;
listen 127.0.0.1;
listen 8000;
listen *:8000;
listen localhost:8000;
```

IPv6 的 IP 地址的设置要使用方括号，例如：

```
listen [::]:8000;
listen [fe80::1];
```

在 Linux 中，通过使用 IPv6 映射地址格式，例如，::ffff:<点分十进制格式的 IPv4 地址>，使得 IPv6 的 TCP 套接字可以接收 IPv4 的流量(就是请求和响应,对于地址来说就是只有流量而言)。例如，::ffff:192.168.0.27 是将 IPv4 的 192.168.0.27 映射到 IPv6 的格式。

如果设置为[::]:80，那么表示将端口 80 绑定到 IPv6 上，在这个 listen 指令中，如果在 Linux 系统中，默认情况下，也启用了 IPv4 下的 80 端口，就是说如果使用了这种格式的设置，那么 Nginx 将会同时监听从 IPv4 和 IPv6 格式的 IP 进入的流量。因此，在使用这种格式的基础上又错误地指定了一个 IPv4 的地址，那么在重新载入 Nginx 的配置时将会有绑定地址的错误发生。

在 Linux 中如果想将 IPv4 和 IPv6 堆栈分开，可以通过运行时参数：net.ipv6.bindv6only 来设置，该参数默认的值为 0，如果想分开使用 IPv4 和 IPv6 套接字，那么使用 sysctl 将该变量设置为 1 即可。

> 注意，任何运行的 Nginx 服务器实例，在改变设置之前将会继续接收 IPv4 的流量。因此，如果需要针对 IPv6 和 IPv4 设置一个新的包处理程序，则需要改变 Nginx 的配置，并且需要重新启动 Nginx 服务器。

另一方面，如果在另一个 Server 上（很明显这里使用的是虚拟主机）仅希望使用 IPv4，例如：

```
listen [::]:80;
```

那么这个 IPv4 绑定将会失败。正确的做法是，在 listen 指令中使用“ipv6only=on”选项，监听 IPv6，而且也可以在格式的 sever 区段指定一个 IPv4 的 listen 指令。

在改变了内核运行时参数之后必须重新编辑配置文件，在这种情况（就是指定的将 IPv4 和 IPv6 套接字分开）下它是最通用的解决方法了。

```
listen [::]:80 ipv6only=on; # listen 指令用于 IPv6，仅监听 IPv6 套接字上的流量。
listen 80;                  # 监听 IPv4 套接字的流量
```

在 FreeBSD 系统中，默认的 IPv4 和 IPv6 的套接字是分开的。“listen [::]:80”仅将 80 端口绑定到 IPv6 套接字，监听 IPv6 的流量，因此，如果想监听 IPv4 的流量，则需要指定 IPv4 的监听指令。

可以在 listen 指令中仅指定 IPv6 地址，通过使用“default_server ipv6only=on”选项来实现，例如：

```
listen [2a02:750:5::123]:80;
listen [::]:80 default_server ipv6only=on;
```

如果仅指定了监听的地址，而没有指定端口，那么 Nginx 将会绑定 80 端口。

下面介绍 default_server 参数。

如果在 listen 指令中使用了 default_server 参数，那么 server{......}区段将成为默认的服务器，对于基于名字的虚拟主机非常有用，可以使用该参数为那些没有匹配 server_name 指定名字的名字提供默认访问。如果没有指定 default_server 参数，则会由第一个 server 提供访问。default_server 参数出现在 0.8.21 版中，而且也不建议继续使用 default 参数。

listen 指令接收一些参数，这些参数用于指定系统调用 listen（2）和 bind（2），而且这些参数必须跟随在 default_server 参数之后。

- backlog=num 参数：该参数用于指定调用 listen（2）时 backlog 的值，默认为-1；
- rcvbuf=size 参数：该参数用于设置监听套接字的 SO_RCVBUF 参数值；
- sndbuf=size 参数：该参数用于设置监听套接字的 SO_SNDBUF 参数值；
- accept_filter=filter 参数：设置过滤器的名字，该参数仅用于 FreeBSD，有两个过滤器——dataready 和 httcodeady；
- deferred 参数：设置该参数则表示延时 accept（2），在 Linux 中使用 TCP_DEFER_ACCEPT 选项；
- bind 参数：该参数指定将 bind（2）分开调用；
- ssl 参数：该参数用于监听 SSL，从 0.7.14 起不再依赖于 listen（2）和 bind（2）的系统调用。例如：

```
listen  80;
listen  443 default_server ssl;
```

例如，使用下面的参数：

```
listen  127.0.0.1 default_server accept_filter=dataready backlog=1024;
```

从 0.8.21 起 Nginx 能够监听 UNIX 套接字，例如：

```
listen unix:/tmp/nginx1.sock;
```

指令名称：location

语法：location [=|~|~*|^~|@] /uri/ { … }

默认值：no

使用环境：server

功能：该指令允许根据 URI 的需要进行配置访问，可以根据字面字符串配置也可以根据正则表达式配置。如果使用正则表达式配置，则必须使用以下的前缀设置。

- “~”区分大小写匹配；
- “~*”不区分大小写匹配。

要决定哪一个 location 指令定义的值能够匹配某一访问，首先是按照字面字符串检测，即按照最明确的匹配查询，然后就是正则表达式检测，对于正则表达式的检测是按照配置文件中的顺序进行检测的。第一个匹配正则表达式的 location 将会被使用，并停止搜索。如果没有正则表达式匹配，那么按照字面字符串搜索到的 location 将会被使用。

指令名称：log_not_found

语法：log_not_found [on|off]

默认值：log_not_found on

使用环境：http，server，location

功能：该指令用于启用或者禁用在磁盘上找不到文件时是否向 error_log 发送日志。

指令名称：log_subrequest

语法：log_subrequest [on|off]

默认值：log_subrequest off

使用环境：http，server，location

功能：该指令用于设置是否在访问日志中启用和禁用子请求。例如，由 rewrite 规则和 SSI 产生的请求，产生的访问日志。

指令名称：msie_padding

语法：msie_padding [on|off]

默认值：msie_padding on

使用环境：http，server，location

功能：该指令用于为 MSIE 和 Chrome 浏览器设置启用或禁用 msie_padding 功能。当启用该指令时，Nginx 将会对小于 512B 的响应体进行填充，填充为 512B。

指令名称：msie_refresh

语法：msie_refresh [on|off]

默认值：msie_refresh off

使用环境：http，server，location

功能：该指令用于允许或者禁止给 MSIE 浏览器请求的响应发送一个 refresh meta 标签。

指令名称：open_file_cache

语法：open_file_cache max = N [inactive = time] | off

默认值：open_file_cache off

使用环境：http，server，location

指令选项：

- Max：该选项用于指定缓存条目的最大值，当还满了后，根据最近最少（LRU）算法将缓存条目移除；
- Inactive：该选项用于指定一个不活动时间，如果在这个时间内缓存的条目没有被访问，那么该条目将会被删除，默认值为 60 秒；
- Off：禁止缓存。

功能：这个指令用于设置是否启用文件缓存。可以缓存的信息如下：

- 打开文件的描述符、文件的大小和修改时间信息；
- 目录存在性的信息；
- 在搜索文件过程中出现的错误信息——找不到文件，没有读取的权限，等等。

例如：

```
open_file_cache max=1000 inactive=20s;
 open file cache valid30s;
 open file cache min uses 2;
 open_file_cache_errors   on;
```

指令名称：open_file_cache_errors

语法：open_file_cache_errors on | off

默认值：open_file_cache_errors off

使用环境：http，server，location

功能：开启或禁用缓存文件错误。

指令名称：open_file_cache_min_uses

语法：open_file_cache_min_uses number

默认值：open_file_cache_min_uses 1

使用环境：http，server，location

功能：该指令用于在指定的时间内一个文件被访问的最少次数，如果访问的次数大于这个值，那么该文件的描述符将会被缓存到缓存中。

指令名称：open_file_cache_valid

语法：open_file_cache_valid time

默认值：open_file_cache_valid 60

使用环境：http，server，location

功能：该指令指定了检测缓存信息的间隔（与 open_file_cache 指令相关）。

指令名称：port_in_redirect

语法：port_in_redirect [on|off]

默认值：port_in_redirect on

使用环境：http，server，location

功能：该指令允许或者阻止端口号在 URL 中的出现。如果 port_in_redirect 设置为 off，那么当请求被重定向时，Nginx 将不会在 URL 中添加端口号。

指令名称：post_action

语法：post_action [uri|off]

默认值：post_action off

使用环境：http，server，location，If-in-location

功能：定义一个请求完成之后的动作。请求完成之后，Nginx 将会调用该 URI。

例如：

```
 location /protected_files {
internal;

proxy pass http://127.0.0.2;
post action /protected done;
 }

 # Send the post_action request to a FastCGI backend for logging.
 location /protected_done {
  internal;
  fastcgi pass 127.0.0.1:9000;
 }
```

指令名称：recursive_error_pages

语法：recursive_error_pages [on|off]

默认值：recursive_error_pages off

使用环境：http，server，location

功能：有的时候，通过指令 error_page 提供的错误网页本身也发生了错误，在这种情况下，指令 error_page 将会被再次使用（递归）。该指令开启或禁用递归错误网页。

指令名称：reset_timedout_connection

语法：reset_timedout_connection [on|off]

默认值：reset_timedout_connection off

使用环境：http，server，location

功能：该指令用于启用或者禁用重新设置连接超时。当一个客户端连接超时，其相关的信息可能仍保留在内存中，依赖于这种状态，相关的信息将依然存在。启用该指令后，如果连接超时，那么将会清除所有与内存的关联。

指令名称：resolver

语法：resolver address

默认值：no

使用环境：http，server，location

功能：该指令用于设置 DNS 服务器。

例如：

```
resolver 127.0.0.1;
```

指令名称：resolver_timeout

语法：resolver_timeout time

默认值：30s

使用环境：http，server，location

功能：域名查询超时时间。

例如：

```
resolver_timeout 5s;
```

指令名称：root

语法：root path

默认值：root html

使用环境：http，server，location，location 中的 If

功能：该指令指定了一个请求的根文档目录。例如下面的配置：

```
location  /i/  {
  root  /spool/w3;
}
```

对于一个“/i/top.gif”的请求，返回的文件将会是“/spool/w3/i/top.gif”。另外，还可以使用变量。

指令名称：satisfy

语法：satisfy [all | any]

默认值：satisfy all

使用环境：http，server，location

功能：该指令定义了客户端是否需要所有访问条件都有效（satisfy all）或者至少一个有效（satisfy any）。

- All：需要所有访问条件都有效；
- Any：至少一个有效。

例如：

```
location / {
  satisfy any;
  allow 192.168.1.0/32;
  deny all;
  auth_basic "closed site";
  auth_basic_user_file conf/htpasswd;
}
```

在上面的例子中，客户能够访问该资源有两个条件。

- 通过 allow 和 deny 指令（HTTP Access 模块），只允许具有本地 IP 的客户端，其他客户端将被拒绝访问；
- 通过 auth_basic 和 auth_basic_user_file 指令（HTTP Auth_basic 模块），只允许能够提供用户名和密码的用户。

使用 satisfy all，客户端必须满足以上两个条件才能够访问该资源；使用 satisfy any，客户端只需满足任意一个条件就可以访问该资源。

指令名称：satisfy_any

语法：satisfy_any [on | off]

默认值：satisfy_any off

使用环境：http，server，location

功能：不建议使用该指令，而使用 satisfy。

指令名称：send_timeout

语法：send_timeout the time

默认值：send_timeout 60

使用环境：http，server，location

功能：设置响应超时，当超过这个设置的时间后，Nginx 将会关闭一个不活动的连接。当一个连接变为非活动状态的那一刻起，客户端便停止传输数据。需要注意的是，这个超时的确定不是整个传输响应的时间，而是两个读操作之间的时间，如果在这个时间内，客户端没有进行任何操作，那么 Nginx 将会关闭连接。

指令名称：sendfile

语法：sendfile [on|off]

默认值：sendfile off

使用环境：http，server，location

功能：该指令用于设置是否使用 sendfile（）。由于这种方法是在内核中进行操作的，因此 sendfile（）比 read（2） 和 write（2）的协作操作更有效，这是由于 read-write 有一个从用户空间传递数据的过程。

指令名称：server

语法：server {...}

默认值：no

使用环境：http

功能：该指令用于配置虚拟主机。

指令名称：server_name

语法：server_name name [...]

默认值：server_name ""

使用环境：server

功能：该指令执行以下两个动作。

- 将进入的 HTTP 请求的主机头（Host header）与 Nginx 配置文件中各个 server { ... }区段比较，并且选择第一个被匹配的 server 区段——这就是如何确定虚拟主机。服务器名字（Server name）按照以下顺序处理：

  ① 全域名，静态域名；

  ② 开始部分使用通配符的域名，例如： *.example.com；

  ③ 结尾部分使用通配符的域名，例如： www.example.*；

  ④ 带有正则表达式的域名。

  如果没有找到匹配的 server，那么会按照下面的顺序在配置文件中选择一个 server { ... }：

  ① 匹配 listen 指令被标记为：[default|default_server]的 server 区段；

  ② 匹配 listen 指令（或者隐含有 listen 80）的第一个 server 区段（或者隐含有 listen80）。

- 如果 server_name_in_redirect 被设置，那么设置用于 HTTP 重定向的服务器名称。

例如：

```
server {
server name   example.com  www.example.com;
}
```

第一个名字成为服务器的基本名称，默认的机器名（hostname）将被使用。可以使用“*”来替代域名的第一部分和最后一部分：

```
server {
   server_name  example.com  *.example.com  www.example.*;
}
```

上面的前两个名字（example.com 和 *.example.com）能够合为一个：

```
server {
server_name  .example.com;
}
```

在服务器名字中使用正则表达式也是可能的，在名字前加上一个波浪符号“~”，像这样：

```
server {
server_name www.example.com  ~^www\d+\.example\.com$;
}
```

从 Nginx 0.7.12 版开始，支持空的服务器名字，能理解（catch）没有“Host”值的头（header）。

> 请注意，大多数浏览器总是发送一个“Host header”，如果使用 IP 访问，那么“Host header”将会包含该 IP。指定一个能够包含所有（catch-all）的访问区段，请参考 listen 指令的 default_server 标志（flag）。

```
server {
  server name "";
}
```

从 Nginx 0.8.25 版本命名捕获（named captures）可以使用在指令 server_name 中：

```
server {
server_name   ~^(www\.)?(?<domain>.+)$;
location / {
root  /sites/$domain;
  }
}
```

一些老版本的 PCRE 提供这种语法，如果有问题发生，那么尝试下列语法：

```
server {
server_name   ~^(www\.)?(?P<domain>.+)$;
location / {
root  /sites/$domain;
}
}
```

指令名称：server_name_in_redirect

语法：server_name_in_redirect on|off

默认值：server_name_in_redirect on

使用环境：http，server，location

功能：该指令应用于内部重定向，如果设置为开启，Nginx 将会使用在 server_name 指令中指定的第一个主机名来做重定向；如果设置为关闭，Nginx 将会使用客户端 HTTP 请求头中 Host 的值做重定向。

指令名称：server_names_hash_max_size

语法：server_names_hash_max_size number

默认值：server_names_hash_max_size 512

使用环境：http

功能：Nginx 使用哈希表（hash table）来做变量数据收集，目的是加速请求进程。该指令用于定义服务器名称哈希表的最大值。如果你的服务器使用的主机名总数超过 512 个，那么你需要增加该值。

指令名称：server_names_hash_bucket_size

语法：server_names_hash_bucket_size number

默认值：server_names_hash_bucket_size 32/64/128

使用环境：http

功能：定义在服务器名称哈希表中一个条目（或者叫记录吧——译者注）的最大长度。如果你的机器名称长度大于 32 个字符，那么你将不得不增加该值。

指令名称：server_tokens

语法：server_tokens on|off

默认值：server_tokens on

使用环境：http，server，location

功能：在错误页面或者是服务器头中是否发生 Nginx 的版本号。

指令名称：tcp_nodelay

语法：tcp_nodelay [on|off]

默认值：tcp_nodelay on

使用环境：http，server，locationn

功能：该指令用于允许或者禁止使用套接字选项 TCP_NODELAY。这个选项只对 keep-alive 连接有效。

指令名称：tcp_nopush

语法：tcp_nopush [on|off]

默认值：tcp_nopush off

使用环境：http，server，location

功能：该指令用于设定允许或者禁用套接字选项 TCP_NOPUSH（在 FreeBSD 系统中）或者 TCP_CORK（在 Linux 系统中），该选项仅在使用 sendfile 时有效。如果 tcp_nopush 设置为 on，那么 Nginx 将会尝试在单个 TCP 数据包中发送整个 HTTP 响应头。

指令名称：try_files

语法：try_files path1 [path2] uri

默认值：none

使用环境：server，location

功能：该指令用于按顺序检测文件的存在性，并且返回第一个找到的文件。$uri/表示是一个目录，即$uri 之后紧跟一个斜线“/”。如果没有找到文件，那么最后的参数——内部重定向将会被使用，最后一个可使用的参数必须存在，否则将会产生内部错误。不像 rewrite，$args 不会自动保存，因此如果最后的参数不是一个命名 location，则需明确指出。

例如：

```
try_files $uri $uri/ /index.php?q=$uri&$args;
```

例如，使用 Mongrel 的代理情况下：

```
    try_files   /system/maintenance.html   $uri   $uri/index.html   $uri.html
@mongrel;

    location @mongrel {
      proxy pass http://mongrel;
    }
```

注意，我们可以指定 HTTP 状态代码，并将其作为最后一个参数，例如：

```
    location / {
      try_files $uri $uri/ /error.php?c=404 =404;
    }
```

在这个设置中，当所有尝试的资源都没有找到时，那么将会由 404 提供。

使用 Drupal/FastCGI 的例子：

```
    # for drupal 6:
    try_files $uri $uri/ @drupal;

    # for drupal 7:
    try files $uri $uri/ /index.php?q=$uri&$args;

    # only needed for Drupal 6 (or if you absolutely need a named location)
    location @drupal {
      rewrite ^ /index.php?q=$uri last; # for drupal 6
    }

    location ~ \.php$ {
      fastcgi pass 127.0.0.1:8888;
      fastcgi param  SCRIPT FILENAME $document root$fastcgi script name;
      # if not already defined in the fastcgi params file
      # any other specific fastcgi_params
    }
```

在这个例子中，try_files 指令：

```
    try_files $uri $uri/ @drupal;
```

级别上类似于：

```
    location / {
      error page 404 = @drupal;
      log not found  off;
    }
```

或者是：

```
    if (!-e $request_filename) {
       rewrite ^ /index.php?q=$uri last;
    }
```

try_files 是对 mod_rewrite 风格的文件/目录存在性检测的一个基本替换，对于 try_files 的使用，其执行效率要比使用 if 更有效。

又如，在 Wordpress 和 Joomla 下使用 try_files：

```
# wordpress (without WP Super Cache) - example 1
try_files $uri $uri/ /index.php?q=$uri&$args;

# wordpress (without WP Super Cache) - example 2
#(it doesn't REALLY need the "q" parameter)
try files $uri $uri/ /index.php;

# joomla
try_files $uri $uri/ /index.php?q=$uri&$args;

location ~ \.php$ {
  fastcgi pass 127.0.0.1:8888;
  fastcgi param  SCRIPT FILENAME $document root$fastcgi script name;
  # if not already defined in the fastcgi params file
  # any other specific fastcgi_params
}
```

指令名称：types

语法：types {...}

使用环境：http，server，location

功能：该指令用设置响应请求文件的文件类型。默认的映射如下：

```
types {
  text/htmlhtml;
  image/gifgif;
  image/jpeg jpg;
}
```

完整的映射表包含在 conf/mime.types 中。

因此，如果想让某一个 location 所有的响应都使用指定的 MIME 类型，例如 application/octet-stream，则可以使用以下设置：

```
location /download/ {
  types { }
  default_type  application/octet-stream;
```

指令名称：underscores_in_headers

语法：underscores_in_headers on|off

默认值：off

使用环境：http，server

功能：允许或者禁止在头中出现下划线。

指令名称：variables_hash_bucket_size

语法：variables_hash_bucket_size size

默认值：variables_hash_bucket_size 64

使用环境：http

功能：定义变量在哈希表中的最大长度，如果变量名大于 64 个字符，则不得不增大该值。

指令名称：variables_hash_max_size

语法：variables_hash_max_size size

默认值：variables_hash_max_size 512

使用环境：http

功能：该指令定义了变量在哈希表中的最大值。如果服务器配置使用的变量总数超过 512 个，那么需要增加该值。

3. 变量

Nginx 的内核模块提供了内置变量，它们的名字与 Apache 的变量名称一致。

变量名称：$arg_PARAMETER

功能：如果在请求中设置了查询字符串，那么这个变量包含在查询字符串是 GET 请求 PARAMETER 中的值。

变量名称：$args

功能：该变量的值是 GET 请求在请求行中的参数。例如，foo=123&bar=blahblah。

变量名称：$binary_remote_addr

功能：二进制格式的客户端地址。

变量名称：$body_bytes_sent

功能：响应体的大小，即使发生了中断或者是放弃，也是一样的准确。

变量名称：$content_length

功能：该变量的值等于请求头中的 Content-length 字段的值。

变量名称：$cookie_COOKIE

功能：该变量的值为 cookie COOKIE 的值。

变量名称：$document_root

功能：该变量的值为当前请求的 location（http，server，location，location 中的 if）中 root 指令中指定的值。

变量名称：$document_uri

功能：同$uri。

变量名称：$host

功能：该变量的值等于请求头中 Host 的值。如果 Host 无效时，那么就是处理该请求的 server 的名称。

在下列情况中，$host 变量的取值不同于$http_host 变量。

- 当请求头中的 Host 字段未指定（使用默认值）或者为空值，那么$host 等于 server_name

指令指定的值。

- 当 Host 字段包含端口号时，$host 并不包含端口号。另外，从 0.8.17 之后的 Nginx 中，$host 的值总是小写。

变量名称：$hostname

功能：由 gethostname 返回值设置机器名。

变量名称：$http_HEADER

功能：该变量的值为 HTTP 请求头 HEADER，具体使用时会转换为小写，并且将“——”（破折号）转换为“_”（下划线）。

变量名称：$is_args

功能：如果设置了$args，那么值为“?”，否则为“”。

变量名称：$limit_rate

功能：该变量允许限制连接速率。

变量名称：$nginx_version

功能：当前运行的 Nginx 的版本号。

变量名称：$query_string

功能：同 $args。

变量名称：$remote_addr

功能：客户端的 IP 地址。

变量名称：$remote_port

功能：客户端的连接断开。

变量名称：$remote_user

功能：该指令的值等于用户的名字，基本身份验证模块使用。

变量名称：$request_filename

功能：该变量等于当前请求文件的路径，由指令 root 或者 alias 和 URI 构成。

变量名称：$request_body

功能：该变量包含了请求体的主要信息。该变量与 proxy_pass 或者 fastcgi_pass 相关。

变量名称：$request_body_file

功能：客户端请求体的临时文件。

变量名称：$request_completion

功能：如果请求成功完成，那么设置为“OK”。如果请求没有完成或者请求不是该请求系列中的最后一部分，那么它的值为空。

变量名称：$request_method

功能：该变量的值通常是 GET 或者 POST。

变量名称：$request_uri

功能：该变量的值等于原始的 URI 请求，就是说从客户端收到的参数包括了原始请求的 URI，该值是不可以被修改的，不包含主机名，例如“/foo/bar.php?arg=baz”。

变量名称：$scheme

功能：该变量表示 HTTP scheme（例如 HTTP，HTTPS），根据实际使用情况来决定，例如：

```
rewrite  ^  $scheme://example.com$uri  redirect;
```

变量名称：$server_addr

功能：该变量的值等于服务器的地址。通常来说，在完成一次系统调用之后就会获取变量的值，为了避开系统调用，那么必须在 listen 指令中使用 bind 参数。

变量名称：$server_name

功能：该变量为 Sever 的名字。

变量名称：$server_port

功能：该变量等于接收请求的端口。

变量名称：$server_protocol

功能：该变量的值为请求协议的值，通常是 HTTP/1.0 或者 HTTP/1.1。

变量名称：$uri

功能：该变量的值等于当前请求中 URI（没有参数，不包括$args）的值。它的值不同于 $request_uri 由浏览器客户端发送的$request_uri 的值。例如，可能会被内部重定向或者是使用 index。

> 另外需要注意，$uri 不包括主机名，例如“/foo/bar.html”。

第 2 章　Nginx 的模块管理和进程管理

Nginx 同 Apache 一样，同样使用了模块化管理，但是与 Apache 有很大的不同，如果说 Apache 支持"热插拔"（就是说如果对 Apache 添加模块不用重新编译 Apache，而只是添加必要的模块，然后再重新载入 Apache 就可以了），那么 Nginx 则必须"重启动"，就是说如果要对 Nginx 服务器添加模块，那么需要重新编译 Nginx 才可以添加相应的功能模块，因此在这一点上要比 Apache 服务器麻烦。

2.1 模块管理

Nginx 采用模块化设计，但它和 Apache 不同的是，模块一旦被编译进来就不可能被卸载，如果有特别需要则只能重新编译 Nginx 了。

2.1.1 从源码看模块

从源代码中可以看出 Nginx 提供的模块：

```
[root@mail nginx-0.8.53]# tree src/
src/
|-- core
|   |-- nginx.c
…
|   |-- ngx_times.c
|   '-- ngx_times.h
|-- event
|   |-- modules
|   |   |-- ngx aio module.c
…
|   |   |-- ngx_select_module.c
|   |   '-- ngx_win32_select_module.c
|   |-- ngx_event.c
…
|   |-- ngx event timer.c
|   '-- ngx event timer.h
|-- http
|   |-- modules
|   |   |-- ngx_http_access_module.c
…
|   |   |-- ngx_http_uwsgi_module.c
|   |   |-- ngx_http_xslt_filter_module.c
|   |   '-- perl
```

```
...
|   |   '-- typemap
|   |-- ngx_http.c
|   |-- ngx_http.h
|   |-- ngx_http_busy_lock.c
|   |-- ngx_http_busy_lock.h
|   |-- ngx_http_cache.h
...
|   |-- ngx_http_variables.h
|   '-- ngx_http_write_filter_module.c
|-- mail
|   |-- ngx_mail.c
...
|   '-- ngx_mail_ssl_module.h
|-- misc
|   |-- ngx_cpp_test_module.cpp
|   '-- ngx_google_perftools_module.c
```

从源码可以看出，Nginx 提供了 core、event、http、mail 和 misc（杂项）五类模块，而每一类模块根据需要又有多种模块，这五类模块中只有 core 模块不能禁用，其他的模块在编译时可以根据实际情况来进行选择。

如同 Apache 一样，Nginx 同样使用了模块化的构建方式，但是与 Apache 不同的是它不支持动态载入模块，即如果在编译安装 Nginx 时没有选择 dav 模块，那么如果想使用它，就只能重新编译 Nginx；同样如果你的 Nginx 在编译安装时选择了 dav 模块，那么现在要想将其去除，也得重新将其编译安装。

Nginx 的发布包中包含了以下模块（以 0.8.53 为例）：

Select、poll、charset、gzip、ssi、userid、access、auth_basic、autoindex、geo、map、split_clients、refere、rewrite、proxy、fastcgi、uwsgi、scgi、memcached、limit_zone、limit_req、empty_gif、browser、upstream、http、http-cache、rtsig、select、poll、aio、ipv6、ss、realip、addition、xslt、image_filter、geoip、http_sub、dav、flv、gzip_static、random_index、secure_link、degradation、stub_status、perl、mail、mail_ssl、mail_pop3、mail_imap、mail_smtp、google_perftools、cpp_test。

Nginx 是模块化结构，因此，在官方提供的版本中有很多模块，在安装时有些模块是可以选择安装的，有些模块是必须安装的，有些模块是默认就安装的，而有些模块在默认安装时却是不被安装的，下面我们以 0.8.53 为例：

默认安装的模块：Select、poll、charset、gzip、ssi、userid、access、auth_basic、autoindex、geo、map、split_clients、refere、rewrite、proxy、fastcgi、uwsgi、scgi、memcached、limit_zone、limit_req、empty_gif、browser、upstream、http、http-cache；

默认没有安装的模块：rtsig、select、poll、aio、ipv6、ss、realip、addition、xslt、image_filter、geoip、http_sub、dav、flv、gzip_static、random_index、secure_link、degradation、stub_status、perl、mail、mail_ssl、mail_pop3、mail_imap、mail_smtp、google_perftools、cpp_test。

看得出，默认安装的模块加上没有默认安装的模块就是 Nginx 发布包中提供的模块。

Nginx 是模块化结构这个没错，但是它和 Apache 不一样，它的模块不能够动态载入或卸载，是一种静态模块系统，当你在编译安装的那一刻就决定了 Nginx 的功能，一旦编译安装完毕，它的功能就已经定了，已安装的模块不能够卸载，而想要使用新的模块，只能重新编译再次安装，这是 Nginx 的结构造成的。其实即使重新编译安装也没多费事，但是一定要记住都使用了前一个版本的哪些功能。

2.1.2　选择使用 Nginx 的模块

在安装 Nginx 时，首先要执行以下命令，以便了解默认安装和默认不安装模块的情况（注意，这是 Nginx-1.0.2 版的情况）：

```
[root@mail nginx-1.0.2]# ./configure --help

  --help this message

…//省略部分

  --with-rtsig_moduleenable rtsig module
  --with-select_module   enable select module
  --without-select_moduledisable select module
  --with-poll_module enable poll module
  --without-poll_module  disable poll module

  --with-file-aioenable file aio support
  --with-ipv6enable ipv6 support

  --with-http_ssl_module enable ngx_http_ssl_module
  --with-http realip module  enable ngx http realip module
  --with-http addition moduleenable ngx http addition module
  --with-http xslt moduleenable ngx http xslt module
  --with-http image filter moduleenable ngx http image filter module
  --with-http_geoip_module   enable ngx_http_geoip_module
  --with-http_sub_module enable ngx_http_sub_module
  --with-http_dav_module enable ngx_http_dav_module
  --with-http_flv_module enable ngx_http_flv_module
  --with-http gzip static module enable ngx http gzip static module
  --with-http random index moduleenable ngx http random index module
  --with-http secure link module enable ngx http secure link module
  --with-http degradation module enable ngx http degradation module
  --with-http_stub_status_module enable ngx_http_stub_status_module

  --without-http_charset_module  disable ngx_http_charset_module
  --without-http gzip module disable ngx http gzip module
  --without-http_ssi_module disable ngx_http_ssi_module
```

```
 --without-http_userid_module disable ngx_http_userid_module
 --without-http_access_module disable ngx_http_access_module
 --without-http_auth_basic_module disable ngx_http_auth_basic_module
 --without-http autoindex moduledisable ngx http autoindex module
 --without-http geo module disable ngx http geo module
 --without-http map module disable ngx http map module
--without-http split clients module disable ngx http split clients module
 --without-http_referer_module disable ngx_http_referer_module
 --without-http_rewrite_module disable ngx_http_rewrite_module
 --without-http_proxy_moduledisable ngx_http_proxy_module
 --without-http_fastcgi_module disable ngx_http_fastcgi_module
 --without-http uwsgi moduledisable ngx http uwsgi module
 --without-http scgi module disable ngx http scgi module
 --without-http memcached moduledisable ngx http memcached module
 --without-http limit zone module disable ngx http limit zone module
 --without-http_limit_req_moduledisable ngx_http_limit_req_module
 --without-http_empty_gif_moduledisable ngx_http_empty_gif_module
 --without-http_browser_module  disable ngx_http_browser_module
 --without-http upstream ip hash module
disable ngx http upstream ip hash module

 --with-http perl moduleenable ngx http perl module
 --with-perl_modules_path=PATH  set path to the perl modules
 --with-perl=PATH   set path to the perl binary

 --http-log-path=PATH   set path to the http access log
 --http-client-body-temp-path=PATH  set path to the http client request body
temporary files
 --http-proxy-temp-path=PATHset path to the http proxy temporary files
 --http-fastcgi-temp-path=PATH  set path to the http fastcgi temporary
files
 --http-uwsgi-temp-path=PATHset path to the http uwsgi temporary files
 --http-scgi-temp-path=PATH set path to the http scgi temporary files

 --without-http disable HTTP server
 --without-http-cache   disable HTTP cache

 --with-mailenable POP3/IMAP4/SMTP proxy module
 --with-mail_ssl_module enable ngx_mail_ssl_module
 --without-mail_pop3_module disable ngx_mail_pop3_module
 --without-mail_imap_module disable ngx_mail_imap_module
 --without-mail smtp module disable ngx mail smtp module
```

```
  --with-google_perftools_module enable ngx_google_perftools_module
  --with-cpp_test_module enable ngx_cpp_test_module

  --add-module=PATH  enable an external module

…//省略
```

在这个帮助信息中，--with-XXX 表示启用，而--without-XXX 则表示禁用，在这里所有--with-XXX 的模块在默认安装时都没有安装，而所有--without-XXX 的模块则是在默认安装时已经被选入的模块，因此，要注意这个规则。也许你已经注意到开始的那四行黑体字，既有--with-XXX 也有--without-XXX，那么这个选择就不是我们选择了，在安装时需根据操作系统的情况来自行决定了。

例如，我们可能会以以下方式来选择安装模块：

```
[root@mail nginx-1.0.2]# ./configure  /
>  --with-http_ssl_module  --with-http_realip_module
>  --without-http_fastcgi_module --without-http_scgi_module
>  --add-module=/root/nginx-accesskey-2.0.3
```

2.1.3 Nginx 使用第三方模块

这个很容易，在对 Nginx 进行 configure 的时候，有一个参数--add-module，就是用来添加第三方模块的，例如 --add-module=/root/nginx-accesskey-2.0.3。

> 这是在“防盗链接”部分的一个例子，可以参考一下该部分内容，以便掌握如何使用--add-module 参数。

2.2 进程管理

Nginx 分为 Single 和 Master 两种进程模型，Single 模型即为单进程方式工作，通过 ngx_single_process_cycle 完成；Master 模型即为一个 master 进程+ n 个 worker 进程的工作方式，通过 ngx_master_process_cycle 完成，可以参考 src/os/unix/ngx_process_cycle.c 源码。

Single 模型顾名思义就是单进程方式工作，容错能力较差，因此不能用于生产之用。而我们的生产环境中都使用的是 master-worker 模型来工作。

因此，在 Nginx 的运行中，会有两种进程存在，那就是 Master 进程和 worker 进程，不同的进程干不同的活。另外，与 Apache 相比，Nginx 使用了更少的资源，算是轻量级的 Web 服务器，如果你的 Nginx 要处理 SSL 或者是 gzip，那么 Nginx 会消耗更多的 CPU 资源。

2.2.1 master 进程和 worker 进程

Nginx 有 master 进程和 worker 进程，即主进程和工作进程，对于它们的区别可以这样来看待：

一个运行在 Linux 2.6 内核的 Nginx，采用的无疑是 epoll 模型，这种模型类似于我们现实生活中的“头负责制”，就是老板与马仔的关系，公司只能有一个老板，他一个人说了算，而可以

有多个马仔，而每个马仔的老板却只有一个，因此形成了“master process > worker process”的关系，老板负责对外，因此他只管接活（揽取业务），而马仔只管干活（处理业务），如果一个马仔的活干不完，那么随后的活就会交给其他的马仔去干（老板就是老板！），因此对于每一个客户端的请求都是由 master 进行接收的，而每一个请求都是由 worker 处理的（没办法，马仔就是马仔！）。在这个模式中，马仔有多个，老板只有一个，老板总是玩命地接活，他总有累死的时候（哈哈，活该！当然也不希望累死老板，树倒猢狲散嘛，也是个麻烦！），那么在这种情况下就得想别的办法，你可能会想到 heartbeat，这个以后再学习。在这里最重要的是理解 master 和 worker 的关系。

对于任何进程都有从开始到消亡的过程，Nginx 的进程也不例外，无论 master 还是 worker。

综上所述，Nginx 服务器和其他 HTTP 服务器一样，它也使用了 master-slave 模型，master 只能有一个，其功能就是在 Nginx 服务器启动时进行全局的初始化（例如根据配置文件进行配置）和管理 worker 进程。

1．master 进程可以处理的信号

master 进程可以处理以下的信号。

信　号	功　能
TERM，INT	快速关闭
USR1	重新打开日志文件
USR2	平滑升级可执行程序
HUP	重新装载配置，在新的工作进程中开始使用新的配置，从容关闭旧的工作进程
WINCH	从容关闭工作进程，对主进程 pid 发送该信号会关闭所有的 worker 进程
QUIT	从容关闭主进程

例如，执行以下命令：

```
[root@cache ~]# ps -ef|grep nginx
root  8387  4590  0 12:09 pts/700:00:00 grep nginx
root  12836  1    0  Aug14  ?00:00:00  nginx:  master  process  nginx  -c
  /usr/local/nginx0.8.53/conf/nginx-varnishd.conf
nobody   12837 12836  0 Aug14 ?00:00:00 nginx: worker process
[root@mail ~]# kill -WINCH 12836
[root@mail ~]# lsof -i:80
COMMANDPID   USER FD   TYPE  DEVICE SIZE NODE NAME
varnishd   2034  nobody   12u  IPv41016199
                      TCP cache.xx.com:60799->localhost:http (CLOSE_WAIT)
nginx  12836 root 6u   IPv4908237   TCP *:http (LISTEN)
```

在这些命令中，我们执行了 **kill -WINCH 12836** 命令，**12836** 为 Nginx 主进程的 PID，但是执行 **lsof -i:80** 命令时，我们会发现，80 端口依然是开启的，无论使用浏览器访问，还是 telnet 命令访问：

```
[root@mail ~]# telnet 192.168.3.139 80
Trying 192.168.3.139...
```

```
Connected to mail.tt.com (192.168.3.139).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com
connection: close
```

都不会返回任何结果，而且会一直持续下去，不会结束访问，也就是断开连接。再次执行 **lsof -i:80** 命令查看：

```
[root@cache ~]# lsof -i:80
COMMANDPID  USER FD  TYPE  DEVICE SIZENODE NAME
varnishd  2034   nobody  12u  IPv4   1016199
          TCP cache.xx.com:60799->192.168.3.140:http (CLOSE_WAIT)
telnet8746   root3u  IPv4 1034340
          TCP cache.xx.com:36331->cache.xx.com:http (ESTABLISHED)
nginx 12836  root6u  IPv4  908237   TCP *:http (LISTEN)
```

现在执行以下命令：

```
[root@cache ~]#kill -HUP 12836
```

在执行命令 kill -HUP 12836 之后，无论是刚才通过浏览器访问的网页还是通过 telnet 访问的网页，立刻就会返回相应的访问页面。

2．worker 进程可以处理的信号

对于 worker 进程，尽管很少对它进行操作，但它同样支持以下信号：

信　号	功　能
TERM，INT	快速关闭子进程
QUIT	从容关闭子进程
USR1	重新打开日志文件

2.2.2　关于 worker 数目的设置

如果你有多个 CPU，那么你可以设置多个 worker，设置的数目可以和 CPU 的核数一样多，Nginx 的作者 Igor Sysoev 是这么说的，可我觉得 worker 的数目似乎要比 CPU 的核数少一个更好（为什么呢？因为 Linux 系统本身要使用 CPU，况且 Nginx 会使用 worker 与 CPU 的亲和力！）。

如果在单个 CPU 上实现多个 worker 进程，那么操作系统会在这多个 worker 之间进行调度，这种情况反而会降低系统的性能，因此，如果是单个 CPU，只设置一个 worker 就可以了。

2.3　针对 Nginx 对 Linux 系统的优化

为了充分发挥 Nginx 服务器的性能，我们需要对其所在的系统进行优化。下面的优化不要完全效仿，而是要根据实际情况进行优化，但大体思想基本一致。

2.3.1 关闭系统中不需要的服务

在刚刚安装完成的 Linux 系统中，总有一些不需要的服务器，需要将其关闭掉，而如果是在已经运行过其他服务器的系统中安装 Nginx 服务器，那么要仔细查看进程，以便找出不用的服务，对于这种情况还需要查找安全问题。因此，如果不是你所运维过的机器或者可以重新安装系统的话，那么就尽可能地重新安装系统。

2.3.2 优化写磁盘操作

我们知道，Nginx 每访问完一个文件之后，Linux 系统将会对它的“Access”，即访问时间进行修改，例如：

```
[root@mail html]# stat  index.html
  File: 'index.html'
  Size: 151 Blocks: 8  IO Block: 4096   regular file
Device: fd00h/64768dInode: 1212214 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-12-01 20:11:47.000000000 +0800
Modify: 2011-12-01 09:22:47.000000000 +0800
Change: 2011-12-01 09:22:47.000000000 +0800
```

通过浏览器访问该文件，然后再看日期：

```
[root@mail html]# stat  index.html
  File: 'index.html'
  Size: 151 Blocks: 8  IO Block: 4096   regular file
Device: fd00h/64768dInode: 1212214 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-12-01 20:12:47.000000000 +0800
Modify: 2011-12-01 09:22:47.000000000 +0800
Change: 2011-12-01 09:22:47.000000000 +0800
```

在一个高并发的访问中，这对磁盘写操作影响是很大的，因此要关闭该功能。

```
/dev/sdb1 /dataext3  defaults  0 0
```

修改为如下配置：

```
/dev/sdb1 /dataext3  defaults,noatime,nodiratime  0 0
```

然后重新启动系统。

如果不能重启系统，那么可以使用 remount 选项来重新挂载：

```
[root@nas ~]# mount -o defaults,noatime, nodiratime -o remount /dev/sdb1 /sdb
[root@nas sdb]# mount |grep sdb1
/dev/sdb1 on /sdb type ext3 (rw,noatime,nodiratime)
```

如果是单独挂载的分区或者磁盘（包括 RAID），可以直接执行以下命令：

```
[root@nas ~]# mount -o  defaults,noatime,nodiratime  /dev/sdb1 /sdb
[root@nas ~]# mount |grep sdb1
/dev/sdb1 on /sdb type ext3 (rw,noatime,nodiratime)
```

2.3.3 优化资源限制

在 Linux 中执行以下两条命令：

```
[root@mail html]# ulimit -n
1024
[root@mail html]# ulimit -u
8040
```

第一条命令，是用于查询单个用户对文件描述符的使用限制，即打开文件的个数，在 Linux 系统中默认值为 1024，即单个用户只能打开 1024 个文件。

第二条命令，是用于查询单个用户最多拥有的进程数，即一个用户所能打开的最大进程数量，在 Linux 系统中默认值是 8040。

对于一个高并发的 Nginx 服务器来说，显然这两个指标值是不够的，因此要对它们进行调整。

调整这两个指标值的方法是通过在 limits.conf 文件中添加以下配置：

```
[root@master ~]# vi /etc/security/limits.conf

  …

* soft nofile 65535
* hard nofile 65535
* soft nproc 65535
* hard nproc 65535
```

需要重新启动系统才会生效。再次查看：

```
[root@master ~]# ulimit -n
65535
[root@master ~]# ulimit -u
65535
```

当然也可以通过 ulimit -a 一起查看。

2.3.4 优化内核 TCP 选项

修改以下 Linux 内核参数：

```
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.tcp tw recycle = 1
net.ipv4.tcp tw reuse = 1
net.ipv4.tcp syncookies = 1
net.core.somaxconn = 262144
net.core.netdev_max_backlog = 262144
net.ipv4.tcp_max_orphans = 262144
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_timestamps = 0
```

```
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 30
```

下面我们来分析一下这些参数。

参数名称：net.ipv4.tcp_max_tw_buckets

默认值：180000。

功能：设置 timewait 的值。

参数名称：net.ipv4.ip_local_port_range

默认值：32768～61000

功能：该参数用于设置允许系统打开的端口范围。

参数名称：net.ipv4.tcp_tw_recycle

默认值：0

功能：该参数用于设置是否启用 timewait 快速回收。

参数名称：net.ipv4.tcp_tw_reuse

默认值：0

功能：该参数用于设置是否开启重新使用，即允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。

参数名称：net.ipv4.tcp_syncookies

默认值：0

功能：该参数用于设置是否开启 SYN Cookies，如果启用该功能，那么当出现 SYN 等待队列溢出时，则使用 Cookies 来处理。

参数名称：net.core.somaxconn

默认值：32768

功能：Web 应用中 listen 函数的 backlog 默认会将内核参数的 net.core.somaxconn 限制到 128，而 Nginx 定义的 NGX_LISTEN_BACKLOG 默认为 511，所以有必要调整这个值。

参数名称：net.core.netdev_max_backlog

默认值：300

功能：该参数用于设置被输送到队列数据包的最大数目，在网卡接收数据包的速率比内核处理数据包的速率快时，那么会出现排队现象，这个参数就是用于设置该队列的大小。

参数名称：net.ipv4.tcp_max_orphans

默认值：32768

功能：该参数用于设置 Linux 能够处理不属于任何进程的套接字数量，所谓不属于任何进程的进程就是“孤儿”（orphan）进程，在快速、大量的连接中这种进程会很多，因此要适当地设置该参数，如果这种“孤儿”进程套接字数量大于这个指定的值，那么在使用 dmesg 查看时会出现“too many of orphaned sockets”的警告。

参数名称：net.ipv4.tcp_max_syn_backlog

默认值：1024

功能：该参数用于记录尚未收到客户端确认信息的连接请求的最大值。

参数名称：net.ipv4.tcp_timestamps

默认值：1

功能：该参数用于设置使用时间戳作为序列号，通过这样的设置可以避免序列号被重复使用。在高速、高并发的环境中，这种情况是存在的，因此通过时间戳能够让这些被看做是异常的数据包被内核接收。将该参数的值设置为 0，表示关闭该功能。

参数名称：net.ipv4.tcp_synack_retries

默认值：5

功能：该参数用于设置 SYN 重试的次数，在 TCP 的 3 次握手中的第 2 次握手，内核需要发送一个回应前面一个 SYN 的 ACK 的 SYN，就是说为了打开对方的连接，内核发出的 SYN 的次数。减小该参数的值有利于避免 DDoS 攻击。

参数名称：net.ipv4.tcp_syn_retries

默认值：5

功能：该参数用于设置在内核放弃建立连接之前发送 SYN 包的数量。

参数名称：net.ipv4.tcp_fin_timeout

默认值：60

功能：表示如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。可以按此设置，但要记住的是，即使是一个轻载的 Web 服务器，也有因为大量的死套接字而内存溢出的风险，FIN-WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能消耗 1.5KB 的内存，但是它的生存期要长些。

参数名称：net.ipv4.tcp_keepalive_time

默认值：7200

功能：当启用 keepalive 的时候，该参数用于设置 TCP 发送 keepalive 消息的频度。

2.4 优化 Nginx 服务器

为了获取更大的性能，有必要对 Nginx 服务器进行优化。

2.4.1 关闭访问日志

关闭 Nginx 的访问日志，如果确实需要记录日志，那么可以根据实际需要有选择地记录部分日志，Nginx 的访问日志可以具体到“区段”级别。

> 在这里我们附加一个说明，本书中提到的“区段”来源于英语的“**context**”，因为在内存的堆栈中是以区段来实现的，而对于 Nginx 指令来说又是用在不同的“**context**”中，因此有时候我们又称之为“使用环境”，实际上是同一个概念的不同叫法。另外，从 Nginx 的配置文件角度来看，还可以叫做“级别”。有的人也叫做“上下文”，我觉的这种叫法不妥，它只不过是按照英文翻译了。
>
> 因此，在本套书（包括卷 1 和卷 2）中提到的“区段”、“级别”和“使用环境”和你在其他地方看到的“上下文”是一个概念。

在 Nginx 中的区段有：http、server、location 等，可以在这些区段中适当地配置日志记录。

2.4.2 使用 epoll

这是在 Linux 下必选的模型，但是 epoll 只能使用于 Linux 内核 2.6 版本及以后的系统。对于我们现在使用的 Linux 系统这不是问题，从 Red Hat 4 以后的系统都是 2.6 内核了。

2.4.3 Nginx 服务器配置优化

下面的指令我们在书中都有分析，这里我们只列出设置它们的值：

- worker_connections 65535
- keepalive_timeout 60
- client_header_buffer_size 8k（通过“getconf PAGESIZE ”命令来获取页面的大小）
- worker_rlimit_nofile 65535

第 3 章　Nginx 如何处理一个请求

Nginx 服务器在处理一个请求时是按照两部分进行的，第一部分是 IP、域名，第二部分是 URI，下面我们将分析这两部分是如何进行工作的。

3.1　IP、域名部分的处理

按照 IP、域名、端口以及 default_server 标志来处理请求。

3.1.1　基于名字的虚拟主机

Nginx 首先决定由哪一个 Server 应该出来处理这个请求。让我们从一个简单的配置文件开始，它包含了三个虚拟主机，均监听在 80 端口：

```
server {
listen 80;
server name  nginx.org  www.nginx.org;
…
}

server {
listen 80;
server name  nginx.net  www.nginx.net;
…
}

server {
listen 80;
server_name  nginx.com  www.nginx.com;
…
}
```

在这个配置中，Nginx 仅测试请求头行中的“Host”，从而来决定将该请求路由到那个 server。如果请求头中的“Host”不与任何服务器名匹配，或者是在请求中根本就没有包含“Host”，那么 Nginx 将会把该请求路由到默认的 server 上。在上面的配置中，默认的 server 是第一个——这是 Nginx 标准的默认行为。如果你不想让列在第一个的 server 作为默认的 server，那么你可以针对需要设置的 server 进行明确设置，设置的方法是：在指令 listen 之后添加参数“default_server”，例如：

```
server {
listen   80  default_server;
```

```
server_name  nginx.net  www.nginx.net;
…
}
```

> "default_server"这个参数是从 0.8.21 版本开始提供的，在早期的版本中，使用的是"default"参数。

> 注意，默认的服务器是针对监听端口来设置的，而不是 server 名字。更详细的内容稍后讲述。

3.1.2　阻止处理对不明确主机名的请求

在客户请求头中，有可能会有 Host 行不明确的情况，如果你不想处理这类用户请求，那么可以定义一个默认的 server 来丢弃这类请求。例如：

```
server {
listen   80  default server;
server name   ;
return   444;
}
```

我们选择一个不存在的域名"_"作为服务器的名字，并且将返回特殊的非标准的代码 444，以便关闭这个连接。

> 需要注意的一点是，应该为这个服务器设置一个名字，否则 Nginx 将会使用它的 hostname。

3.1.3　基于 IP 和域名的虚拟域名服务器处理请求

让我们来看一个比较复杂的配置，在这个配置文件中有若干虚拟主机监听在不同的 IP 上：

```
server {
listen   192.168.1.1:80;
server_name  nginx.org  www.nginx.org;
…
}

server {
listen   192.168.1.1:80;
server_name  nginx.net  www.nginx.net;
…
}

server {
listen   192.168.1.2:80;
server_name  nginx.com  www.nginx.com;
```

```
    …
    }
```

在这个配置中，Nginx 首先测试与 server 区段“listen”指令对应的 IP 地址和端口号，然后再测试与 server 区段 server_name 指令值对应的请求头中 Host 行中的值，即 IP 优先，如果没有找到相应的服务器名字，那么该请求将会由默认的服务器进行处理。

如前所述，默认服务器是指令 listen 监听端口的一个属性，并且不同的默认服务器可以被定义在不同的监听端口：

```
server {
listen192.168.1.1:80;
server name   nginx.org  www.nginx.org;
…
}

server {
listen192.168.1.1:80  default_server;
server_name   nginx.net  www.nginx.net;
…
}

server {
listen192.168.1.2:80  default_server;
server_name   nginx.com  www.nginx.com;
…
}
```

3.2 URI 部分的处理

在 URI 部分是通过 location 来实现的，可以使用正则表达式。

3.2.1 实例

让我们来看一个典型的、简单的 PHP 网站 Nginx 是如何选择一个 location 处理一个请求的：

```
server {
listen80;
server name   nginx.org  www.nginx.org;
root  /data/www;

location / {
index index.html  index.php;
}

location ~* \. (gif|jpg|png) $ {
```

```
expires 30d;
}

location ~ \.php$ {
fastcgi_pass   localhost:9000;
fastcgi_param  SCRIPT_FILENAME
   $document_root$fastcgi_script_name;
includefastcgi_params;
}
}
```

Nginx 首先通过字面字符串逐字搜索给定的、最明确的 location 区段，而不管它们在配置文件中列举的顺序。

在上面的配置中，仅有的字面 location 是“/”，由于它可以匹配任何请求，因此它将被作为最后使用（也可以说是最后的一种补救办法），然后会按照配置文件中的顺序，Nginx 开始检查通过正则表达式给定的 location，第一个匹配的表达式将会停止本次的搜索，Nginx 就会使用该 location。如果没有正则表达式匹配这个请求，那么 Nginx 会使用先前找到的最明确的字面字符的 location。

注意，所有类型的 location 测试仅需要请求的 URI 部分而不包括查询字符串。这么做是因为查询字符串可以在各自的方式中给定，例如：

```
/index.php?user=john&page=1
/index.php?page=1&user=john
```

此外，任何人都可以请求在查询字符串中的任何字符：

```
/index.php?page=1&something+else&user=john
```

3.2.2 分析

下面让我们来看一下在上面的配置文件中 Nginx 是如何处理请求的。

- 对于一个“/logo.gif”的请求将会被具有明确字符的 location“/”首先匹配，然后就是正则表达式“\.（gif|jpg|png）$”，因此，该请求将会被后面的 location 处理。由于使用了指令“root /data/www”，因此该请求的文件被映射到一个“/data/www/logo.gif”文件上，并且将该文件发送到了客户端。
- 对于一个“/index.php”的请求，同样会被具有明确字符的 location“/”首先匹配，然后是正则表达式“\.（php）$”，因此，该请求将会被后面的 location 处理，并且将该请求传递到监听在 localhost:9000 的 FastCGI 服务器。指令“fastcgi_param”将 FastCGI 参数 SCRIPT_FILENAME 设置为“/data/www/index.php”，并且 FastCGI 服务器会执行该文件。变量$document_root 的值等于指令“root”的值，变量$fastcgi_script_name 的值等于请求的 URI，例如“/index.php”。
- 对于一个“/about.html”的请求，它仅会被具有明确字符的 location“/”匹配，因此，它会被这个 location 处理。使用了“root /data/www”指令将请求的文件匹配到“/data/www/about.html”，然后再发送到客户端。
- 处理一个“/”比较复杂，它仅与 location“/”匹配，因此，它也只能由这个 location 处

理。由指令 index 根据它的参数和指令“root /data/www”来测试一个 index 文件的存在性，然后由指令 index 做一个内部重定向到“/index.php”，并且 Nginx 会再次搜索该 location，并将请求发送到客户端。正如我们前面看到的，该被重定向的请求最终将会由 FastCGI 服务器处理。

第 4 章　服务器名字

服务器的名字是由指令 server_name 来定义，并且也决定了使用哪一个 server 区段来提供对客户端请求的响应。服务器名字的定义可以使用准确的名字（exact name）、通配符名字（wildcard name）或者是正则表达式:

```
server {
listen   80;
server name  nginx.org  www.nginx.org;
...
}

server {
listen   80;
server_name  *.nginx.org;
...
}

server {
listen   80;
server_name  mail.*;
...
}

server {
listen   80;
server_name  ~^(?<user>.+)\.nginx\.net$;
...
}
```

这些名字的测试顺序为:

- 准确的名字；
- 以星号开始的通配符名字: *.nginx.org;
- 以星号结尾的通配符名字: mail.*；
- 按照正则表达列举在配置文件中的顺序。

在这四种情况中第一个匹配后就停止搜索。

4.1 通配符名字

通配符名字仅可以在名字的开始或结尾包含一个星号，星号仅在点“.”号的边缘。像下列

名字，“www.*.nginx.org”和“w*.nginx.org”均为无效名字。然而，这些名字能够通过使用正则表达式来指定，例如，“~^www\..+\.nginx\.org$”或“~^w.*\.nginx\.org$”。星号能够匹配名字的一些部分，例如，名字“*.nginx.org”不但能够匹配 www.nginx.org，而且还能够匹配 www.sub.nginx.org。

一个特殊的通配符格式“.nginx.org”，不但能够准确匹配名字“nginx.org”，而且能够匹配通配符名字“*.nginx.org”。

4.2 正则表达式名字

要使用 Nginx 提供的正则表达式名字，那么在编译安装 Nginx 时必须首先安装 Perl 编程语言正则表达式（PCRE）。为了使用正则表达式，在服务器名字开始之前使用一个波浪号字符“~”：

```
server_name  ~^www\d+\.nginx\.net$;
```

否则，就会被作为准确的名字来对待，或者是如果在表达式中包含一个星号（*），那么就会被作为一个通配符名字（最有可能成为无效的名字）。不要忘记设置锚符号“^”和“$”，它们不需要在语法上，而是在逻辑上。

> 同样要注意的是，在域名中的点号“.”要使用反斜线进行转义。

另外，一个包含有“{”和“}”的正则表达式需要使用引号：

```
server_name  "~^(?<name>\w\d{1,3}+)\.nginx\.net$";
```

否则，Nginx 将会在启动时失败，并显示以下错误信息：

```
directive "server_name" is not terminated by ";" in ...
```

被捕获的命名正则表达式（named regular expression）在以后可以作为变量：

```
server {
server_name   ~^(www\.)?(?<domain>.+)$;

location / {
root   /sites/$domain;
}
}
```

PCRE 库支持命名捕获（named capture），遵循下列语法。

- ?<name>：兼容 Perl 5.10 语法，从 PCRE-7.0 开始支持。
- ?'name'：兼容 Perl 5.10 语法，从 PCRE-7.0 开始支持。
- ?P<name>：Python 语法兼容，从 PCRE-4.0 开始支持。

如果 Nginx 启动失败并显示以下错误消息：

```
pcre_compile() failed: unrecognized character after (?< in ...
```

这个消息的意思是说 PCRE 库太旧，你可以尝试一下语法“?P<name>”。

另外在使用捕获时，也可使用数字形式：

```
server {
server_name   ~^(www\.)?(.+)$;

location / {
root   /sites/$2;
}
}
```

然而，这种使用方法仅限于简单的情况（如上述），因为数字的引用能够很容易地被覆盖。

4.3 其他不同种类的名字

如果你想通过不是默认的 server 区段来处理一个在请求头行（header line）中却没有包含“Host”的请求，那么你应该指定一个空的名字：

```
server {
listen   80;
server name  nginx.org  www.nginx.org  "";
…
}
```

如果客户端使用了一个没有在 server 区段通过 server_name 指定的名字，那么 Nginx 会使用一个空的名字作为服务器的名字。

> 在这种情况下，如果使用的 Nginx 是 0.8.48 以上的版本，那么会使用主机名（hostname）作为服务器名字。

如果一个客户端的请求使用了 IP 地址，而不是服务器名字，那么在请求头行（header line）中，“Host”包含的就不是服务器名字，而是 IP 地址，那么在这种情况下，如果想让客户端通过 IP 访问到某个 server 区段，那么可以在 Nginx 的配置文件指定的 server 区段中指定适当的 IP 地址：

```
server {
listen   80;
server_name  nginx.org
 www.nginx.org
 ""
 192.168.1.1
 ;
…
}
```

在捕获所有服务器的例子中，你可能会看到一个奇怪的名字“_”：

```
server {
listen   80  default_server;
```

```
    server_name  _;
    return  444;
    }
```

这里指定的不是什么特别的名字，它只是一个无效的域名，从来不会与任何真实名字相匹配。你也可以使用类似于“--”，“!@#”等符号。

在 Nginx 中，高于 0.6.25 的版本会支持一个特别的名字“*”，这个特别的名字会匹配所有解释错误的服务器名字。它从来不会担任起包括所有服务器名字或者是通配符服务器名字，事实上，它提供的功能已由指令“server_name_in_redirect”提供，特殊名字“*”现在已不提倡使用，而应该使用指令“server_name_in_redirect”。

> 注意，绝对不能使用指令“server_name”来指定获取所有服务器名字（catch-all name）或者是否（default）服务器，这个是“listen”的属性，而不是指令“server_name”的属性。

可以参考“Nginx 如何处理一个请求（How nginx processes a request）”，可以定义服务器监听在端口*:80 和 *:8080，而直接将默认的服务器端口设定为*:8080，而*:80 为默认端口：

```
    server {
    listen  80;
    listen  8080  default server;
    server name  nginx.net;
    …
    }

    server {
    listen  80  default_server;
    listen  8080;
    server name  nginx.org;
    …
    }
```

4.4 名字优化

准确的名字和通配符名字作为哈希值被存储在哈希表中，这些哈希值被绑定到监听端口上，每一个监听的端口有三个哈希值：

- 一个准确名字的哈希值；
- 一个由星号开始名字的哈希值；
- 一个由星号结尾名字的哈希值。

该哈希值大小的优化配置分阶段进行，因此，在 CPU 的缓存中在最少失误的情况下找到该名字。

准确名字的哈希值首先被搜索；如果一个名字没有通过准确名字的哈希值找到，那么将会使

用由星号开始名字的哈希值搜索，如果仍然没有搜索到，那么会使用由星号结尾名字的哈希值搜索。需要理解的一点是——搜索通配符名字哈希值要慢于搜索准确名字哈希值，这是由于名字是通过域名部分搜索所致。

注意，通配符的特殊形式“.nginx.org”是被存储在通配符名字哈希表中，而不是存储在准确名字哈希表中；如果仍然没有找到，那么就会继续使用正则表达式的方法测试查找，理论上这是一种最慢的方法，并且也不能扩展。

基于这些原因，最好尽可能使用确切名称。例如，如果一台服务器被请求最多的名字是 nginx.org 和 www.nginx.org，那么更高效率的定义是将它们明确规定：

```
server {
listen   80;
server_name  nginx.org  www.nginx.org  *.nginx.org;
…
}
```

而不要使用这种简单的格式：

```
server {
listen   80;
server_name  .nginx.org;
…
}
```

如果你定义了大量的服务器名字，或者是定义了少见的长的服务器名字，那么你需要在 http 级别（或者叫区段）调整指令“server_names_hash_max_size”和“server_names_hash_bucket_size”的值。指令“server_names_hash_bucket_size”默认值可以是 32、64 或者是其他值，这依赖于 CPU 缓存行（ache line）的大小。如果默认值为 32，当你定义“too.long.server.name.nginx.org”作为服务器的名字，那么 Nginx 将会在启动时失败，并且显示如下错误信息：

```
could not build the server_names_hash,
you should increase server_names_hash_bucket_size: 32
```

在以下情况时，应该设置该指令的值为以前值的两倍：

```
http {
server names hash bucket size  64;
…
```

如果你指定了大量的服务器名字，那么将会遇到另一个错误信息：

```
could not build the server_names_hash,
you should increase either server names hash max size: 512
or server_names_hash_bucket_size: 32
```

应该首先尝试设置“server_names_hash_max_size”的值接近于服务器名字的数量。如果修改这个指令的值仍然没有起到作用，或者是当 Nginx 的启动太慢时，才去尝试增加指令“server_names_hash_bucket_size”的值。如果服务器仅监听在一个端口，那么 Nginx 根本就不会检测服务器的名字（也不会为该端口建立哈希表）。然而，却有一个例外，如果指令

“server_name”的值是一个捕获的正则表达式，那么 Nginx 不得不执行一个表达式来获取捕获。

4.5 兼容性

- 从 0.8.48 版本开始，可以使用空名字来表示默认服务器。
- 从 0.8.25 版本开始，可以使用命名正则表达式（named regular expression）来捕获服务器名字。
- 从 0.7.40 版本开始，支持正则表达式服务器名字捕获。
- 从 0.7.12 版本开始，支持空“”服务器名字。
- 从 0.6.25 版本开始，通配符服务器名字或正则表达式名字被作为首选的服务器名字来使用。
- 从 0.6.7 版本开始，支持正则表达式。
- 从 0.6.0 版本开始，支持通配符格式“nginx.*”。
- 从 0.3.18 版本开始，支持特殊格式“.nginx.org”。
- 从 0.1.13 版本开始，支持通配符格式“*.nginx.org”。

4.6 对服务器名字的扩展

通过正则表达式实现其他的一些“服务器名字”。

4.7 基于目录名的域名访问

4.7.1 正则表达式处于主机名字的位置上

```
[root@mail sites-enabled]# vi mail.t1.com

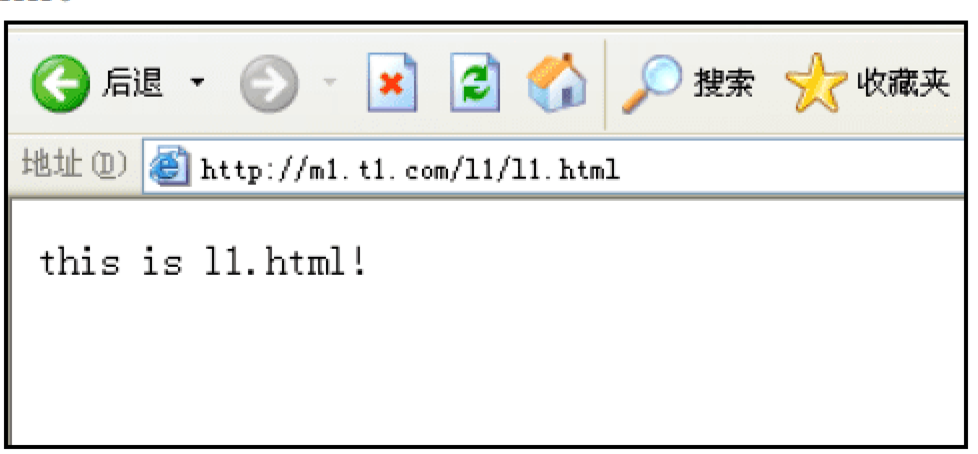
server {
listen   80;
server_name  ~^(.+)?\.t1\.com$;
root  /usr/local/nginx0.8/html/t1.com/$1;
location   / {
index  index.html index.htm;
}
}
```

在这里我们使用了正则表达式域名，域名已定，主机名可以随便，但是主机名不要太长，否则也会出现其他的问题。在这里从理论上来说可以访问 xxx.t1.com，其中 xxx 为目录/usr/local/nginx0.8/html/t1.com 下的任何目录名，例如：

```
[root@mail html]# tree t1.com/
t1.com/
|-- index.html
|-- m1
|   |-- index.html
|   '-- l1
|   '-- l1.html
|-- m2
|   '-- index.html
|-- m3
|   '-- index.html
'-- m4
'-- index.html
```

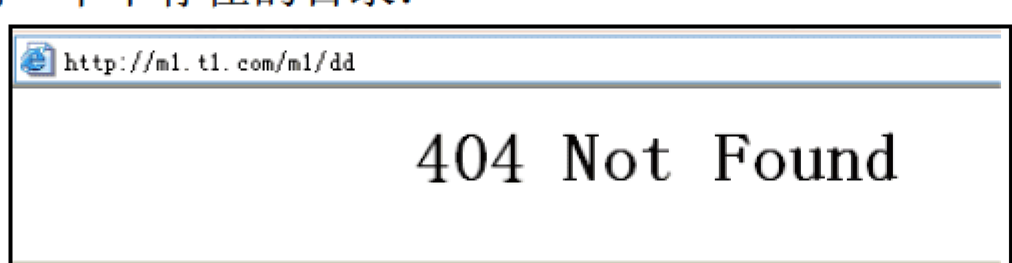
这样，我们的目录名可以随便地扩展，在访问时，使用“目录名”+“域名”即可，都不用重新载入配置文件或重启服务器。例如：m1.t1.com、m2.t1.com 及 m3.t1.com 等。

不用担心二级子目录的访问，对于 m1/l1/l1.html，我们的访问方法同样是：http://m1.t1.com/l1/l1.html。



所以说嘛，肯定是没问题的！

我们看一下对于域名的解析情况，由于访问存在的网页/目录不会在错误日志中记录任何信息，因此在这里我们访问一个不存在的目录：



我们查看错误日志的情况：

```
2010/12/05 16:22:50 [error] 920#0: *32 open() "/usr/local/nginx0.8/html/t1.com/m1/dd" failed (2: No such file or directory), client: 192.168.3.248, server: ~^(.+)?\.t1\.com$, request: "GET /dd HTTP/1.1", host: "m1.t1.com"
```

在这里注意三点：

- “**usr/local/nginx0.8/html/t1.com/m1/dd**”，这是由客户端请求 http://m1.t1.com/m1/dd 转变而来；
- 响应客户端请求的服务器名字“**server: ~^（.+）?\.t1\.com$**”；
- 客户端的请求被解析的主机名“**host: "m1.t1.com"**”。可见客户端的访问能够正确地被分解到文档的位置上。

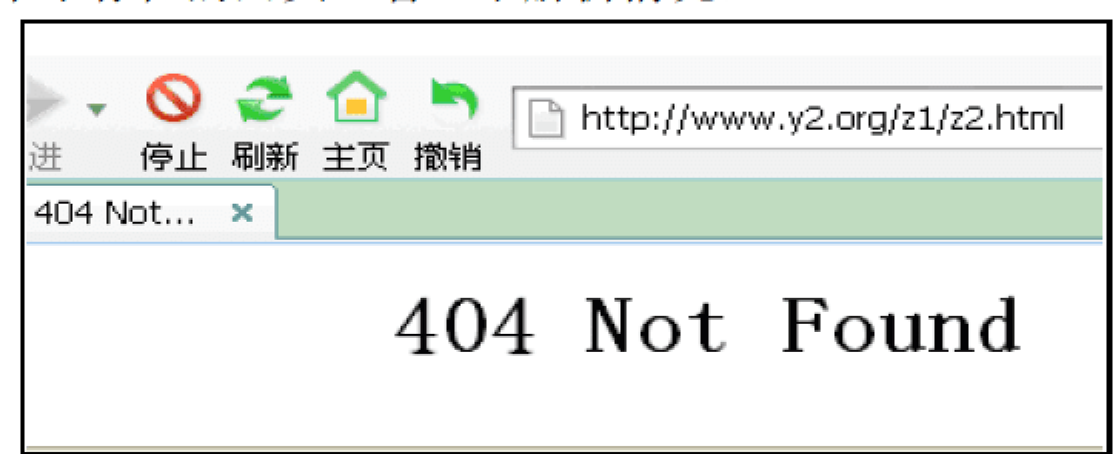
4.7.2 正则表达式处于域名的位置上

```
[root@mail html]# tree www
www
|-- index.html
|-- y1.cn
|   '-- index.html
|-- y1.com
|   '-- index.html
|-- y1.org
|   '-- index.html
|-- y2.cn
|   '-- index.html
|-- y2.com
|   '-- index.html
'-- y2.org
|-- index.html
'-- z1
'-- z1.html
```

这个例子是一个主机名为 www，而域名可以是任意合法的域名（但是必须注册！）。同上面一样，我们来进行访问：



在上面的截图中，我们访问了 http://www.y2.org/ 和 http://www.y2.org/z1/z1.html 都是没问题的。下面访问一个不存在的网页，看一下解析情况：



查看出错日志情况：

```
2010/12/06 08:57:24 [error] 18912#0: *68 open() "/usr/local/nginx0.8/html/www/y2.org/z1/z2.html" failed (2: No such file or directory), client: 192.168.3.248, server: ~^(www\.)?(.+)$, request: "GET /z1/z2.html HTTP/1.1", host: "www.y2.org"
```

同样要注意三点：

- “ **/usr/local/nginx0.8/html/www/y2.org/z1/z2.html** ”，这是由客户端请求 http://m1.t1.com/m1/dd 转变而来；
- 响应客户端请求的服务器名字“**server: ~^（www\.）?（.+）$**”；
- 客户端的请求被解析的主机名“**host: "www.y2.org"**”。可见此时接受客户端访问的 Server 已经该变了。

问题

如果此时再去访问 m1.t1.com，就会出现：



查找出错的原因，从错误日志着手：

```
2010/12/06 09:27:45 [error] 20104#0: *2 "/usr/local/nginx0.8/html/www/m1.t1.com/index.html" is not found (2: No such file or directory), client: 100.100.170.248, server: ~^(www\.)?(.+)$, request: "GET / HTTP/1.1", host: "m1.t1.com"
```

同样注意上面提到的三点：

- 客户访问的域名“**m1.t1.com**”；
- 接受访问的 Server；
- 被解析后访问文档的路径“**/usr/local/nginx0.8/html/www/m1.t1.com/index.html**”。

分析出错的原因：在于客户端服务的“**m1.t1.com**”应该由 **server: ~^（.+）?\.t1\.com$**来解析，但实际却是由 **server: ~^（www\.）?（.+）$**解析，因此导致了访问文档路径的不正确。

解决方法

这是修改后的虚拟主机配置文件：

```
[root@mail sites-enabled]# ls
mail.t1.com  mail.t2.com
[root@mail sites-enabled]# more  *
::::::::::::::
mail.t1.com
::::::::::::::
server {
listen   100.100.150.139:80  ;
server_name  ~^(.+)?\.t1\.com$;
root  /usr/local/nginx0.8/html/t1.com/$1;
location   / {
index  index.html index.htm;
}
}
```

```
::::::::::::::
mail.t2.com
::::::::::::::
server
  {
listen   192.168.3.139:80;
server_name  ~^(www\.)?(.+)$;
index index.php index.html;
root  /usr/local/nginx0.8/html/www/$2;
location  / {
index  index.html index.htm;
}
}
```

分别把这两个正则表达式域名绑定在了两个不同的 IP 上，当然在 DNS 解析时也需要这样处理。然后就可以正确地访问了。下面访问了两个网页，一个是“m1.t1.com”，另一个是“m1.t1.com/ff”，其中第二个是不存在的，目的是看一下错误日志：



可见对于“m1.t1.com”的访问已经正确地转换，通过“m1.t1.com/ff”的出错来证实一下：

```
2010/12/06 09:41:31 [error] 20150#0: *2 open() "/usr/local/nginx0.8/html/
t1.com/m1/dd" failed (2: No such file or directory), client: 100.100.170.248,
server: ~^(.+)?\.t1\.com$, request: "GET /dd HTTP/1.1", host: "m1.t1.com"
```

同样注意上述三点：

- 客户访问的域名“**m1.t1.com**”；
- 接受访问的 **Server**；
- 被解析后访问文档的路径“**/usr/local/nginx0.8/html/t1.com/m1/dd**”。

通过上面的日志能够看出，对于域名“m1.t1.com”的访问，已被正确地解析到 server: ~^(.+)?\.t1\.com$，客户端访问的文档也能够被正确地分解到磁盘的位置。

4.8 关于$1、$2…的使用

在上面的配置中我们使用了$1、$2，如果将“root /usr/local/nginx0.8/html/www/$2”中的$2 改为$1，那么则会出现以下情况：

```
2010/12/06 10:00:49 [error] 20652#0: *4 "/usr/local/nginx0.8/html/www/www./index.html" is not found (2: No such file or directory), client: 192.168.3.248, server: ~^(www\.)?(.+)$, request: "GET / HTTP/1.1", host: "www.y2.org"
```

注意“**/usr/local/nginx0.8/html/www/www./index.html**”中，在解析中出现了 **www.**，这说明 Nginx 在转换过程中已经将“**server: ~^（www\.）?（.+）$**”中的“**www.**”作为变量$1 的值了，而其余的部分作为了$2 的值。

第 5 章　协助用户操作 Nginx 的工具

为了更好地使用 Nginx 服务器，我们可以使用下面的这些工具。这些工具能够帮助我们更加有效、直观和方便地管理 Nginx 服务器。

5.1 工具 1——nginx.vim

nginx.vim 是一个辅助工具，通过下面的配置，在使用时它将成为 vim 工具的一部分，具体的作用是用于编辑 nginx 的配置文件。在没有使用该工具之前，我们先看一下在使用 vim 编辑 Nginx 配置文件的情况：

```
server {
    listen       192.168.3.139:80;
    server_name  mail.tt.com;
    return       403;
    }

server {
    listen       192.168.3.139:80   ;
    server_name  mail.t1.com;
   return       402;
    }
server {
    listen       100.100.150.139:80   ;
    server_name  mail.t2.com;
    return       405;
    }
```

如上图所示，无论是指令还是参数均为白色字。

下面我们下载并设置该工具，然后再查看其结果。

5.1.1 下载与安装

设定目录并下载 nginx.vim 文件：

```
[root@mail ~]# mkdir -p  ~/.vim/syntax/
[root@mail ~]# wget http://www.vim.org/scripts/download_script.php?src_id=14376
```

编辑 filetype.vim 文件：

```
[root@mail ~]# vi ~/.vim/filetype.vim
au BufRead,BufNewFile /usr/local/nginx0.8/conf/* set ft=nginx
```

5.1.2 使用

在 filetype.vim 文件中添加以上内容。需要注意的一点是，要正确地指明 Nginx 配置文件所在的位置，例如 在这里是“/usr/local/nginx0.8/conf/*”，根据实际情况修改这里就可以了。

再次对该配置文件进行编辑，你会比较满意：

```
server {
    listen       192.168.3.139:80;
    server_name  mail.tt.com;
    return       403;
    }

server {
    listen       192.168.3.139:80  ;
    server_name  mail.t1.com;
   return       402;
    }
server {
    listen       100.100.150.139:80  ;
    server_name  mail.t2.com;
    return       405;
    }
```

由于黑白印刷，文字颜色看不清楚，实际上，上图中代码颜色是不同的，大家可以实际操作一下。通过这个辅助工具能够使你在配置中自觉地注意配置命令的正确性，这对于初学者会有很大的帮助。该工具由 Evan Miller 编写，我们表示感谢。

5.2 工具 2——eperusio-nginx_ensit

eperusio-nginx_ensit 是一个 shell（Bash）脚本，是 Debian 系统下用于控制 Apache 2.2 虚拟主机命令 a2ensite 和 a2dissite 的复制版，用于控制 Nginx。

原始的 a2ensite 和 a2dissite 用 Perl 语言编写，a2dissite 是 a2ensite 的一个符号链接，在本工具中，开发者遵循了同样的方法，例如，nginx_dissite 是 nginx_ensite 的一个符号链接。

5.2.1 下载与安装

下载：

```
wget http://download.github.com/perusio-nginx_ensite-f410035.tar.gz
```

解压：

```
[root@mail ~]# tar -zxvf perusio-nginx ensite-f410035.tar.gz
perusio-nginx ensite-f410035/
perusio-nginx_ensite-f410035/README.org
perusio-nginx_ensite-f410035/bash_completion.d/
perusio-nginx_ensite-f410035/bash_completion.d/nginx-ensite
perusio-nginx_ensite-f410035/man/
perusio-nginx ensite-f410035/man/nginx dissite.8
perusio-nginx ensite-f410035/man/nginx ensite.8
perusio-nginx ensite-f410035/nginx dissite
perusio-nginx ensite-f410035/nginx ensite
perusio-nginx_ensite-f410035/nginx_ensite.sig
```

为了便于了解，查看一下它的目录结构：

```
[root@mail ~]# tree perusio-nginx ensite-f410035
perusio-nginx ensite-f410035
|-- README.org
```

```
|-- bash_completion.d
|   '-- nginx-ensite
|-- man
|   |-- nginx dissite.8
|   '-- nginx ensite.8
|-- nginx dissite -> nginx ensite
|-- nginx ensite
'-- nginx_ensite.sig
```

首先了解一下包内有何物：

- README.org：这个文件其实不用多说，但一定要读（如果你真正地把本文看懂了，你就可以继续往下看了）；
- bash_completion.d：这是一个唯一的目录，该目录中包含了一个脚本文件；
- nginx-ensite：该脚本文件的作用不是很大，因此不再详述，有兴趣的用户可以自己看看；
- man：该目录下的两个文件是相应的 nginx_dissite 和 nginx_ensite 帮助文档，在后面会将其翻译为中文的；
- nginx_ensit：用于启动虚拟主机的具体命令；
- nginx_dissite：用于关闭虚拟主机的具体命令，可以看得出，它仅是 nginx_ensit 命令的一个链接；
- nginx_ensite.sig：签名文件，用于安全验证。

了解完包内的文件后，明白了只有脚本（nginx_ensite）及其符号链接（nginx_dissite）是必需的。因此，只需要将该脚本（nginx_ensite）及其符号链接（nginx_dissite）丢到/usr/sbin目录中或者是系统上的其他位置，意思就是：cp nginx_* /usr/sbin，就是这样，你便大功告成了。如果你想查看一下 man 文档，那么还要执行以下的操作。为了操作简单，修剪了一下目录：

```
[root@mail ~]# mv perusio-nginx ensite-f410035 perusio-nginx
[root@mail ~]# cd perusio-nginx
[root@mail perusio-nginx]#cp nginx_* /usr/sbin  //这样复制会变成两个文件
[root@mail perusio-nginx]#ll /usr/sbin/nginx *
-rwxr-xr-x 1 root root 3722 Dec  2 17:25 /usr/sbin/nginx_dissite
-rwxr-xr-x 1 root root 3724 Dec  2 17:33 /usr/sbin/nginx_ensite
[root@mail perusio-nginx]#cp man/* /usr/local/share/man/man8
```

或者是：

```
[root@mail perusio-nginx]#cp man/* /usr/share/man/man8
```

这样就可以使用 man 命令来查看相关命令的用法了。

5.2.2 相关命令

下面学习两个命令的用法：

1. 命令

nginx_ensite，nginx_dissite：开启或者是关闭一个 Nginx 站点/虚拟主机。

2. 用法

```
nginx_ensite [site]
```

或：

```
nginx_dissite [site]
```

3. 描述

简单地描述了命令 nginx_ensite 和 nginx_dissite 的用法。

nginx_ensite 是一个脚本，它能够启动一个包含在 Nginx 配置文件 http 区段的虚拟主机，通过在目录/etc/nginx/sites-enabled 中建立符号链接来实现。同样，nginx_dissite 能够停止一个虚拟主机，是通过删除一个链接来实现。启动一个已经启动的站点或者是关闭一个已经关闭的站点都不会报错（或者说是被认为属于正确的操作）。

对于默认的站点会被特殊处理：它的链接被叫做“000-default”，这是为了它被首先载入。

例如：

```
nginx_dissite  default
```

该命令将会关闭默认站点。

相关文件及目录：

```
/etc/nginx/sites-available
```

该目录下的文件都是有效站点的配置文件。

```
/etc/nginx/sites-enabled
```

该目录下的配置文件均为目录 sites-available 下文件的链接，本目录的链接数少于或等于（少于是说明有的站点被停用了，等于就不用说了吧！）目录 sites-available 下的文件数，这些链接都是用于启动相应站点（虚拟主机）的配置文件。

> 请注意，该脚本为 Nginx 的配置假定了一个明确的文件系统的拓扑结构，因此在具体的使用时要么修改 Nginx 的配置，要么修改该脚本的假设，或者对二者都做适当的修改，以便适合使用。

所有的虚拟主机配置文件（virtual hosts configuration files）都应该放置在目录/etc/nginx/sites-available 下，例如，虚拟主机 foobar 的配置放置在一个文件中，然后再将该文件放置在目录/etc/nginx/sites-available 中；

运行脚本 nginx_ensite，并且将 foobar 作为参数：nginx_ensite foobar，这样一个符号链接/etc/nginx/sites-enabled/foobar -> /etc/nginx/sites-available/foobar 就被建立了；

该脚本使用了 nginx -t 来测试配置文件是否正确，如果测试失败，那么符号链接不会建立，还会输出错误提示；

如果所有的配置都是正确的，那么现在就会重新载入 Nginx，在基于 Debian 的系统中，这意味着执行/etc/init.d/nginx reload；

好了，现在是见证奇迹的时刻了，点击你的浏览器，然后去访问新配置的主机（当然是虚拟主机的域名了！），假设你的配置是合理的，你会发现一切都正常工作。要禁用一个站点，只需要运行 nginx_dissite foobar 即可，该脚本会重新载入 Nginx 来更新运行环境。

5.2.3 实例

本实例中，Nginx 安装在/usr/local/nginx0.8/目录中，当然配置文件所在的命令就是/usr/local/nginx0.8/conf/了。

完成了上面的安装步骤后——就是将 nginx_ensite 和 nginx_dissite 脚本复制到/usr/sbin下，接着进行下面的步骤：

（1）建立 sites-enabled 和 sites-available 目录：

```
[root@mail conf]# pwd
/usr/local/nginx0.8/conf
[root@mail conf]# mkdir sites-available
[root@mail conf]# mkdir sites-enabled
```

（2）编辑虚拟主机配置文件：

```
[root@mail conf]# cd sites-available
[root@mail sites-available]# ls//根据需要编辑以下三个虚拟主机文件
mail.t1.com  mail.t2.com  mail.t3.com
[root@mail sites-available]# more *
::::::::::::::
mail.t1.com
::::::::::::::
server {
listen   80  ;
server name  mail.t1.com;
root  /usr/local/nginx0.8/html/t1;
location   / {
index  index.html index.htm;
}

}
::::::::::::::
mail.t2.com
::::::::::::::
server {
listen   80;
server name  mail.t2.com;
root  /usr/local/nginx0.8/html/t2;
location  / {
index  index.html index.htm;
}

}
::::::::::::::
mail.t3.com
```

```
::::::::::::::
server {
listen   80;
server name  mail.t3.com;
root  /usr/local/nginx0.8/html/t3;
location / {
index  index.html index.htm;
}


}
```

需要明确指出的是，在这里我们将所有虚拟主机配置文件放置在 sites-available 目录中。

（3）编辑 nginx.conf 文件，使用 include 命令载入虚拟主机配置文件：

```
[root@mail conf]# vi nginx.conf

worker_processes  1;

events {
worker connections  1024;
}

http {
include   mime.types;
default_type  application/octet-stream;
sendfileon;
keepalive timeout  65;

include "sites-enabled/mail*";

}
```

在这里需要注意一点，命令 include 的参数是 **sites-enabled** 而不是 sites-available。

（4）修改命令 nginx_ensite、nginx_dissite 中的部分配置。

修改命令 nginx_ensite：

```
[root@mail ~]# cat  /usr/sbin/nginx_ensite
#!/bin/bash
# nginx ensite --- Bash script to enable or disable a site in nginx.
…

NGINX=$ (which nginx)

## The paths for both nginx configuration files and the sites
```

```
## configuration files and symbolic link destinations.
#NGINX_CONF_DIR=/etc/nginx
NGINX_CONF_DIR=/usr/local/nginx0.8/conf    //修改后
AVAILABLE SITES PATH="$NGINX CONF DIR/sites-available"
ENABLED SITES PATH="$NGINX CONF DIR/sites-enabled"

SCRIPTNAME=${0##*/}

## Checking the type of action we will perform. Enabling or disabling.
…

## action if possible. If not signal an error and exit.
case $ACTION in
ENABLE)
if [ -r $SITE AVAILABLE ]; then
  [ -h $SITE_ENABLED ] || ln -s $SITE_AVAILABLE $SITE_ENABLED
  # Test for a well formed configuration.
  echo "Testing nginx configuration..."
  $NGINX -t && STATUS=0
  if [ $STATUS ]; then
  /etc/init.d/nginx reload    //添加该行
  echo -n "site $1 has been enabled."
  echo "Run /etc/init.d/nginx reload to apply the changes."
  exit 0
  else
  exit 2
  fi
else
echo "Site configuration file $1 not found."
exit 3
fi
;;
DISABLE)

…

esac
```

修改命令 nginx_dissite：

```
[root@mail ~]# cat  /usr/sbin/nginx_dissite
#!/bin/bash

# nginx ensite --- Bash script to enable or disable a site in nginx.
```

```
…

## The paths for both nginx configuration files and the sites
## configuration files and symbolic link destinations.
#NGINX CONF DIR=/etc/nginx
NGINX_CONF_DIR=/usr/local/nginx0.8/conf   //修改后
AVAILABLE SITES PATH="$NGINX CONF DIR/sites-available"
ENABLED_SITES_PATH="$NGINX_CONF_DIR/sites-enabled"

…
## action if possible. If not signal an error and exit.
case $ACTION in
ENABLE)
if [ -r $SITE AVAILABLE ]; then
  [ -h $SITE ENABLED ] || ln -s $SITE AVAILABLE $SITE ENABLED

…
;;
DISABLE)
if [ -h $SITE ENABLED ]; then
rm $SITE ENABLED
echo "Site $1 has been disabled."
/etc/init.d/nginx reload                    //添加该行
echo "Run /etc/init.d/nginx reload to apply the changes."
exit 0
else
echo "Site $1 doesn't exist."
exit 3
fi
;;
esac
```

上面已经说明了：一定要根据实际情况进行修改。

（5）解决相关问题。

你注意到“NGINX=$（which nginx）”没有？在这里我有两个可能假设：一是如果安装有多个 Nginx；二是是否已在你的 PATH 变量中。

有两个假设就有两个问题（但愿你哪个都碰不上，你的运气一直都不错！）需要解决。先说第一个，如果在你的系统中安装有多个 Nginx，那么你就不要使用“NGINX=$（which nginx）”了，而是直接写入路径就可以了。

例如，“NGINX= /usr/local/nginx0.8/sbin/nginx”。

对于第二个问题，你可以编辑“/etc/profile”，然后找到 PATH 部分：

```
[root@mail sites-available]# vi  /etc/profile
```

```
# /etc/profile
…
PATH=$PATH:$JAVA_HOME/bin:$JAVA_HOME/jre/bin:/usr/local/nginx0.8/sbin
CLASSPATH=.:$JAVA HOME/lib
export PATH JAVA_HOME CLASSPATH
```

在 PATH 的尾部添加上“usr/local/nginx0.8/sbin”。

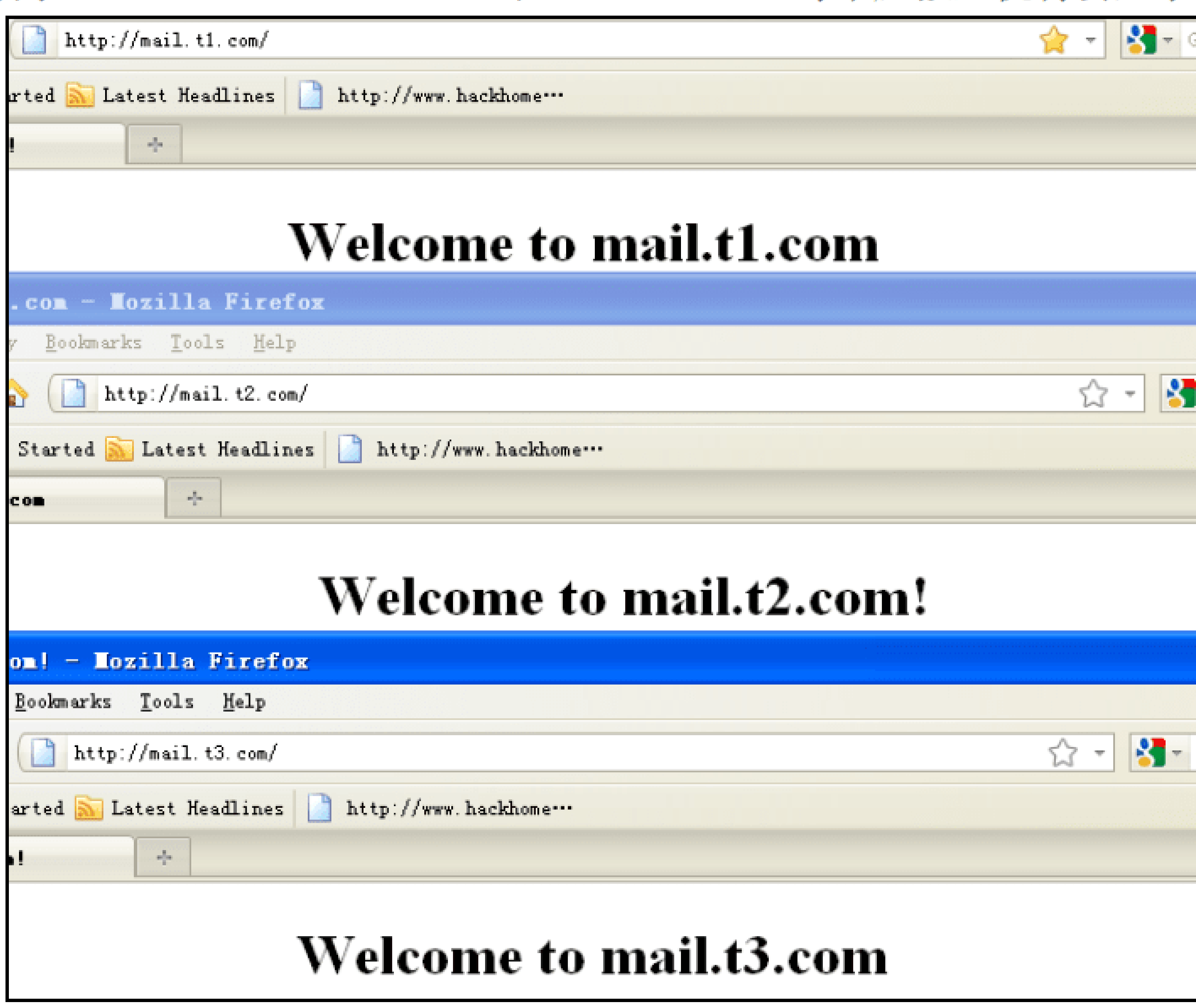
（6）将每个要启动的虚拟主机作为链接：

```
[root@mail sites-available]# nginx_ensite mail.t1.com
[root@mail sites-available]# nginx_ensite mail.t2.com
[root@mail sites-available]# nginx_ensite mail.t3.com
```

每个虚拟主机配置文件的链接由 nginx_ensite 命令自动来完成。每添加一个虚拟主机都要重新载入配置文件，因此会出现类似以下的信息：

```
[root@mail ~]# nginx ensite mail.t1.com
Testing nginx configuration...
the configuration file /usr/local/nginx0.8/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx0.8/conf/nginx.conf test is successful
the configuration file /usr/local/nginx0.8/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx0.8/conf/nginx.conf test is successful
Reloading nginx:   [  OK  ]
site mail.t1.com has been enabled.Run /etc/init.d/nginx reload to apply the
  changes.
```

（7）依次访问 mail.t1.com、mail.t2.com 和 mail.t3.com，每个虚拟主机将会显示自己的主页：

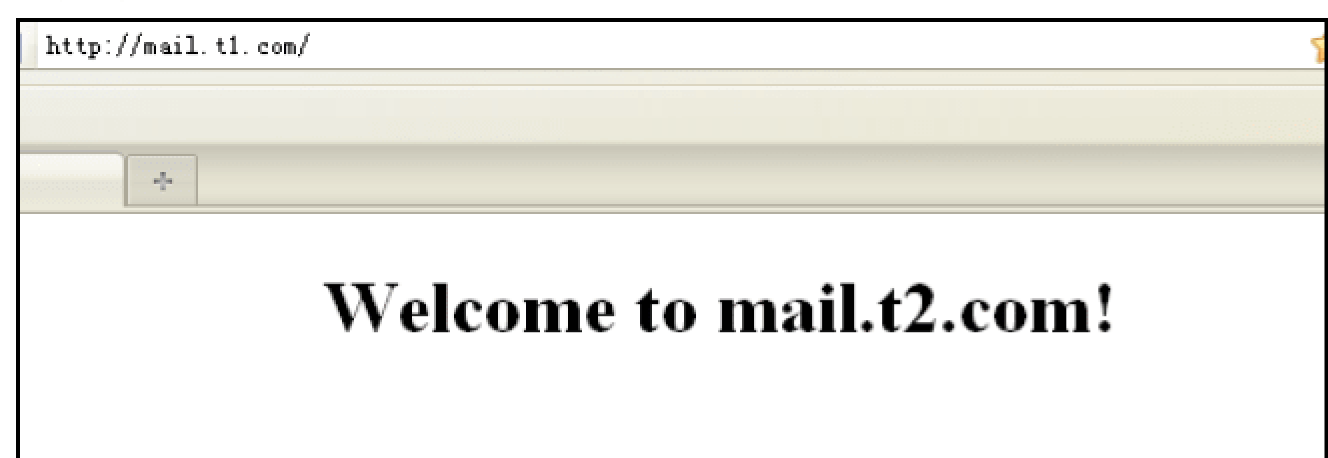


（8）执行 nginx_dissite 命令，关闭需要关闭的虚拟主机：

```
[root@mail ~]#nginx dissite mail.t1.com
Site mail.t1.com has been disabled.
```

```
the configuration file /usr/local/nginx0.8/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx0.8/conf/nginx.conf test is successful
Reloading nginx:  [  OK  ]
Run /etc/init.d/nginx reload to apply the changes..
```

然后再次访问 mail.t1.com：

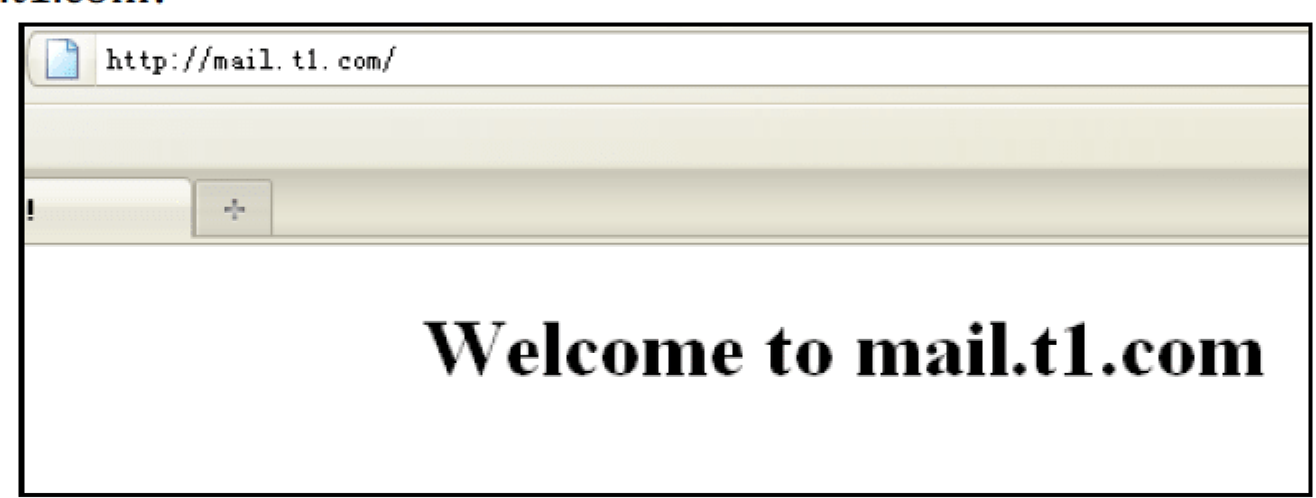


可以看得出，访问被默认主机提供（为什么默认主机是 mail.t1.com 呢？这个可以参考“服务器名字部分”）。

为了证实这个工具的可用性，我们再执行以下命令：

```
[root@mail ~]#nginx ensite mail.t1.com
Testing nginx configuration...
the configuration file /usr/local/nginx0.8/conf/nginx.conf syntax is ok
configuration file /usr/local/nginx0.8/conf/nginx.conf test is successful
site mail.t1.com has been enabled.Run /etc/init.d/nginx reload to apply the changes.
```

再次访问 mail.t1.com：



又回到了 mail.t1.com 的主页。

该工具由 perusio 编写，如果你觉得有用，那么你去感谢他吧！

5.3 工具 3——htpasswd.py

如果没有安装 Apache，那么可以使用 htpasswd.py 具来产生和管理 htpasswd 文件。

编写作者是 Eli Carter，我们同样对作者表示感谢！

5.3.1 下载文件

```
#!/usr/bin/python
"""Replacement for htpasswd"""
```

```
# Original author: Eli Carter

import os
import sys
import random
from optparse import OptionParser

# We need a crypt module, but Windows doesn't have one by default.  Try to
  find
# one, and tell the user if we can't.
try:
import crypt
except ImportError:
try:
import fcrypt as crypt
except ImportError:
sys.stderr.write ("Cannot find a crypt module.  "
 "Possibly http://carey.geek.nz/code/python-fcrypt/\n")
sys.exit (1)


def salt () :
"""Returns a string of 2 randome letters"""
letters = 'abcdefghijklmnopqrstuvwxyz' \
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ' \
  '0123456789/.'
return random.choice (letters)  + random.choice (letters)


class HtpasswdFile:
"""A class for manipulating htpasswd files."""

def __init__ (self, filename, create=False) :
self.entries = []
self.filename = filename
if not create:
if os.path.exists (self.filename) :
self.load ()
else:
raise Exception ("%s does not exist" % self.filename)

def load (self) :
"""Read the htpasswd file into memory."""
```

```
lines = open (self.filename, 'r') .readlines ()
self.entries = []
for line in lines:
username, pwhash = line.split (':')
entry = [username, pwhash.rstrip () ]
self.entries.append (entry)

def save (self) :
"""Write the htpasswd file to disk"""
open (self.filename, 'w') .writelines (["%s:%s\n" % (entry[0], entry[1])
 for entry in self.entries])

def update (self, username, password) :
"""Replace the entry for the given user, or add it if new."""
pwhash = crypt.crypt (password, salt () )
matching_entries = [entry for entry in self.entries
if entry[0] == username]
if matching_entries:
matching entries[0][1] = pwhash
else:
self.entries.append ([username, pwhash])

def delete (self, username) :
"""Remove the entry for the given user."""
self.entries = [entry for entry in self.entries
if entry[0] != username]


def main () :
"""%prog [-c] -b filename username password
Create or update an htpasswd file"""
# For now, we only care about the use cases that affect tests/functional.py
parser = OptionParser (usage=main.__doc__)
parser.add option('-b', action='store true', dest='batch', default=False,
help='Batch mode; password is passed on the command line IN THE CLEAR.'
)
parser.add_option   (   '-c',    action='store_true',    dest='create',
  default=False,
help='Create a new htpasswd file, overwriting any existing file.')
parser.add_option ('-D', action='store_true', dest='delete_user',
default=False, help='Remove the given user from the password file.')

options, args = parser.parse_args ()
```

```
def syntax_error (msg) :
"""Utility function for displaying fatal error messages with usage
help.
"""
sys.stderr.write ("Syntax error: " + msg)
sys.stderr.write (parser.get_usage () )
sys.exit (1)

if not options.batch:
syntax_error ("Only batch mode is supported\n")

# Non-option arguments
if len (args)  < 2:
syntax_error ("Insufficient number of arguments.\n")
filename, username = args[:2]
if options.delete_user:
if len (args)  != 2:
syntax error ("Incorrect number of arguments.\n")
password = None
else:
if len (args)  != 3:
syntax_error ("Incorrect number of arguments.\n")
password = args[2]

passwdfile = HtpasswdFile (filename, create=options.create)

if options.delete user:
passwdfile.delete (username)
else:
passwdfile.update (username, password)

passwdfile.save ()


if   name   == '  main  ':
main ()
```

5.3.2 命令的使用方法

将上面的代码保存为文件 httppasswd，放在合适的位置下。下面看一下该工具的使用方法：

```
[root@mail ~]# python httppasswd -h
usage: httppasswd [-c] -b filename username password
```

```
    Create or update an htpasswd file

    options:
      -h, --help  show this help message and exit
      -b  Batch mode; password is passed on the command line IN THE CLEAR.
      -c  Create a new htpasswd file, overwriting any existing file.
      -D  Remove the given user from the password file.
```

毫无疑问，它的功能就是建立或更新 htpasswd 文件。它提供了三个有用的选项，命令格式也比较简单，但可以满足我们的需要。

5.4 工具 4——Nginx 启动脚本

由于在默认的安装包中没有提供管理脚本，为了管理方便，因此我们需要为它添加一个启动/关闭/脚本。它是一个 shell 脚本文件。在这里需要注意的是，要根据自己的安装情况做必要的修改。在下面的代码中，需要修改的地方都用加粗字体标出来了。

```
[root@mail ~]# more /etc/init.d/nginx
#!/bin/sh
#
# nginx - this script starts and stops the nginx daemon
#
# chkconfig:   - 85 15
# description:  Nginx is an HTTP(S) server, HTTP(S) reverse \
#   proxy and IMAP/POP3 proxy server
# processname: nginx
# config:  /usr/local/nginx0.8/conf/nginx.conf
# config:  /etc/sysconfig/nginx
# pidfile: /var/run/nginx.pid

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ "$NETWORKING" = "no" ] && exit 0

nginx="/usr/local/nginx0.8/sbin/nginx"
prog=$(basename $nginx)

NGINX_CONF_FILE="/usr/local/nginx0.8/conf/nginx.conf"
```

```
[ -f /etc/sysconfig/nginx ] && . /etc/sysconfig/nginx

lockfile=/var/lock/subsys/nginx

make_dirs() {
   # make required directories
   user='nginx -V 2>&1 | grep "configure arguments:" | sed 's/[^*]*--user=\
   ([^ ]*\).*/\1/g' -'
   options='$nginx -V 2>&1 | grep 'configure arguments:''
   for opt in $options; do
   if ['echo $opt | grep '.*-temp-path'' ]; then
   value='echo $opt | cut -d "=" -f 2'
   if [ ! -d "$value" ]; then
   # echo "creating" $value
   mkdir -p $value && chown -R $user $value
   fi
   fi
   done
}

start() {
[ -x $nginx ] || exit 5
[ -f $NGINX_CONF_FILE ] || exit 6
make_dirs
echo -n $"Starting $prog: "
daemon $nginx -c $NGINX_CONF_FILE
retval=$?
echo
[ $retval -eq 0 ] && touch $lockfile
return $retval
}

stop() {
echo -n $"Stopping $prog: "
killproc $prog -QUIT
retval=$?
echo
[ $retval -eq 0 ] && rm -f $lockfile
return $retval
}

restart() {
configtest || return $?
```

```
stop
sleep 1
start
}

reload() {
configtest || return $?
echo -n $"Reloading $prog: "
killproc $nginx -HUP
RETVAL=$?
echo
}

force_reload() {
restart
}

configtest() {
  $nginx -t -c $NGINX CONF FILE
}

rh_status() {
status $prog
}

rh_status_q() {
rh status >/dev/null 2>&1
}

case "$1" in
start)
rh_status_q && exit 0
$1
;;
stop)
rh status q || exit 0
$1
;;
restart|configtest)
$1
;;
reload)
rh_status_q || exit 7
```

```
$1
;;
force-reload)
force reload
;;
status)
rh status
;;
condrestart|try-restart)
rh_status_q || exit 0
;;
*)
echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart|reload|
  force-reload|configtest}"
exit 2
esac
```

具体使用该脚本时，只要适当地修改标出的部分就可以了。

第 6 章　5xx 错误及处理

使用 Nginx 做代理，由于后端的服务器发生故障，php-cgi 进程数不够用、php 执行时间长或者是 php-cgi 进程死掉，以及 Nginx 端 FastCGI 缓存使用情况（如果你使用的是代理，那么要注意代理缓存的使用情况），在这些情况下都会出现 502、504 错误，总是报“502 bad gateway”、“504 gateway time-out”有不愿意让最终访问者看到这个故障，而是想利用其他的网页掩饰一下，比如说“服务器正在维护，请您稍后访问！！！”等比较委婉的语言。

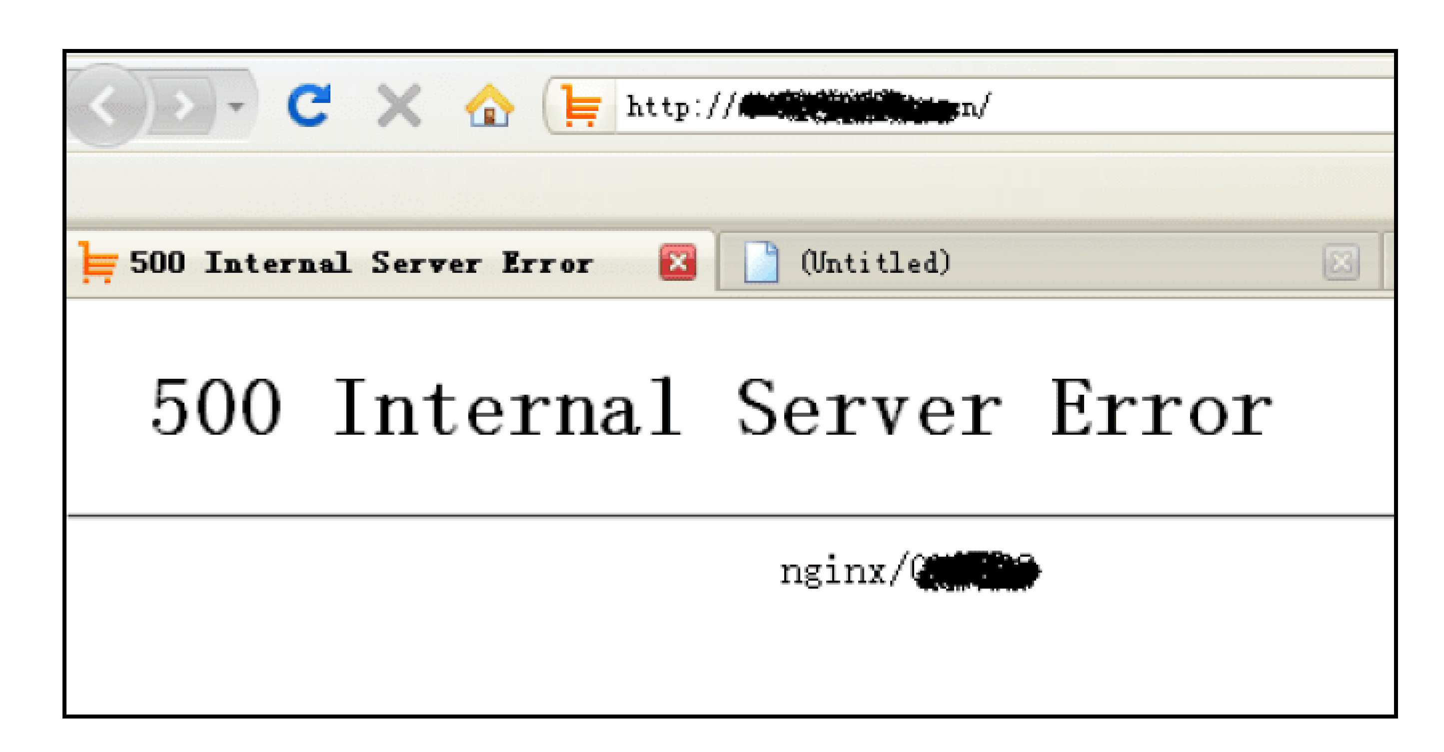
Nginx 所报告的“502 Bad Gateway”指定的是请求 PHP-fpm 已经执行，但是由于某种原因（基本上是读取资源的问题）而没有完全执行完毕，最终导致 PHP-fpm 进程终止。

Nginx 所报告的“504Gateway Time-out”的含义指定的是客户端所发出的请求没有到达网关，换句话说就是请求没有到可以执行的 PHP-fpm。

一般来说，Nginx 报告的“502 Bad Gateway”错误和 php-fpm.conf 的设置有关，而 Nginx 报告的“504 Gateway Time-out”则是与 nginx.conf 的设置有关。

6.1　500 内部服务器错误

在一台 Nginx 服务器上最近频繁发生 500 错误，尤其是在访问量大的时候，如下图所示。



6.1.1　问题分析

根据 HTTP 协议的内容分析，500 为内部服务器错误，即服务器遇到意外情况而无法履行请求。于是查看 Nginx 的错误日志：

```
[root@s17 logs]# tail -f  nginx_error.log

2011/06/20 16:35:35 [alert] 4208#0: *913163 socket () failed (24: Too many
   open files) while connecting to upstream, client: 125.75.8
1.90,  server:  weed.xx.cn,  URL:  "/assets/js/jquery/jquery.client.js",
```

```
upstream: "http://192.168.4.11:80/assets/js/jquery/jquery.
    client.js", host: "weed.xx.cn", referrer: "http://happy.xx.cn/346/c/
    201106/20/n3378729,20.shtml"
    2011/06/20 16:35:35 [alert] 4208#0: *913166 socket () failed (24: Too many
      open files) while connecting to upstream, client: 218.21.2
    42.58,server:jhad.xx.cn,URL:"/tad.jsp?id=50",upstream: "http://192.
      168.4.10:8080/jhad/tad.jsp?id=50", host: "jhad.xx.
    cn", referrer: "http://happy.xx.cn/346/c/201106/20/n3378729,21.shtml"
    2011/06/20 16:35:35 [alert] 4208#0: *913080 socket () failed (24: Too many
      open files) while connecting to upstream, client: 60.223.2
    45.201, server: happy.xx.cn, URL: "/lib/js/jquery/jquery-1.4.2.min.js",
upstream: "http://192.168.4.8:80/lib//js/jquery/jquery-
    1.4.2.min.js", host: "happy.xx.cn", referrer: "http://happy.xx.cn/346/c/
    201106/20/n3378729,20.shtml
```

看上面节选日志中的黑体字部分，说明是由于超过了最大打开文件数的限制。通过下面的命令查看原有文件描述符可用的情况：

```
[[root@s17 logs]# ulimit -n
10240
```

6.1.2 问题解决

为了解决问题，可以考虑两种方法，一种是在 Nginx 的配置文件中进行修改；而另一种则是在操作系统的级别上做修改。本人认为还是在 Nginx 配置文件级别上进行修改为妥。

在 Nginx 配置文件级别上的修改：

```
[root@s17 conf]# vi nginx.conf

user  www www;
worker processes 4;
worker_rlimit_nofile 65535;
error_log logs/nginx_error.log  crit;
events
{
use epoll;
worker connections 10240;
}
```

黑体字部分是我们添加的配置，添加后需要重新载入 Nginx 的配置才能生效。

在操作系统级别上的修改：

在操作系统级别上修改的方法是通过修改文件/etc/security/limits.conf 的配置来完成，在该文件中添加以下两条配置语句：

```
[root@s17 logs]# vi /etc/security/limits.conf
```

```
…

* soft nofile 65535
* hard nofile 65535
```

将上述语句添加到文件的尾部即可。这种方法需要重新启动系统。下面再检查一下：

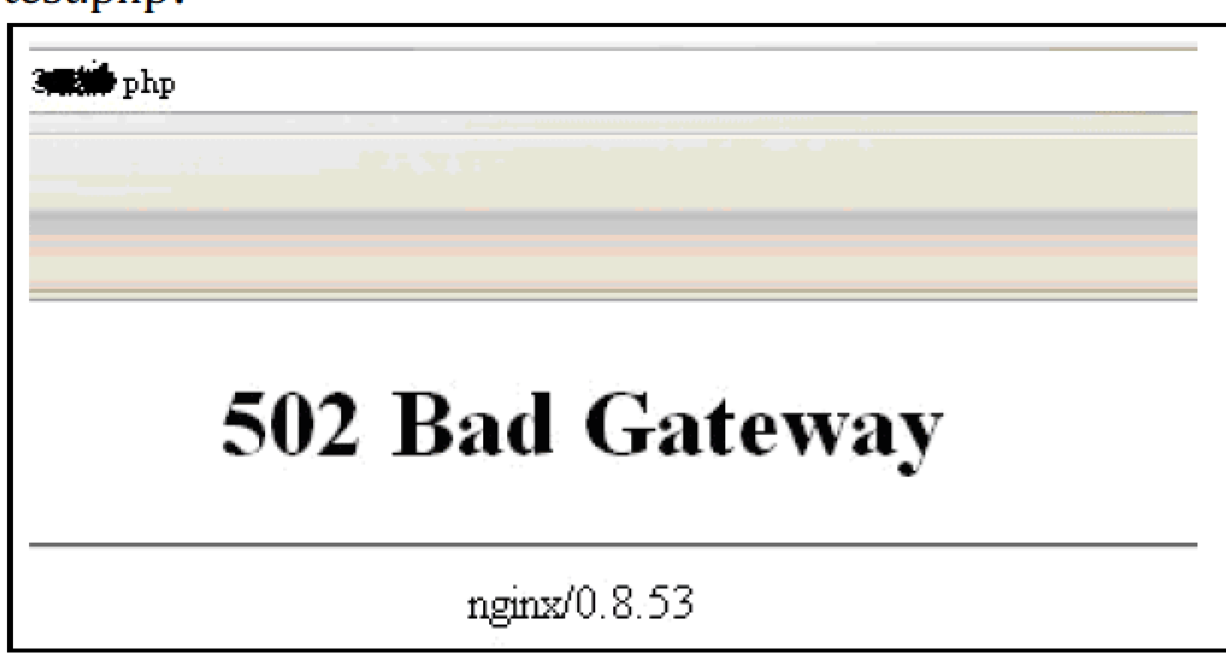
```
[root@s17 logs]# ulimit  -n
65535
```

没错，是这样的。

通过这两种方法的修改都能够解决“500 Internal Server Error”问题，但是如果访问继续增大，超出了 65535 的限制，只好考虑其他的解决方法了，因为这个方法已经到达操作系统的极限了，如果碰到了可以考虑集群。

6.2 502 问题—— 502 bad gateway

在没有修改 Nginx 配置之前，我们先来访问一下。首先停掉 php-fpm，访问 http://192.168.3.25/test.php：



显然不是我们想要的。

第一种方法：使用 **error_page** 命令。

针对这种情况，下面做一个示范。首先查看修改后的 Nginx 配置文件：

```
http {
include   mime.types;
default_type  application/octet-stream;
sendfileon;
keepalive_timeout  5;

…

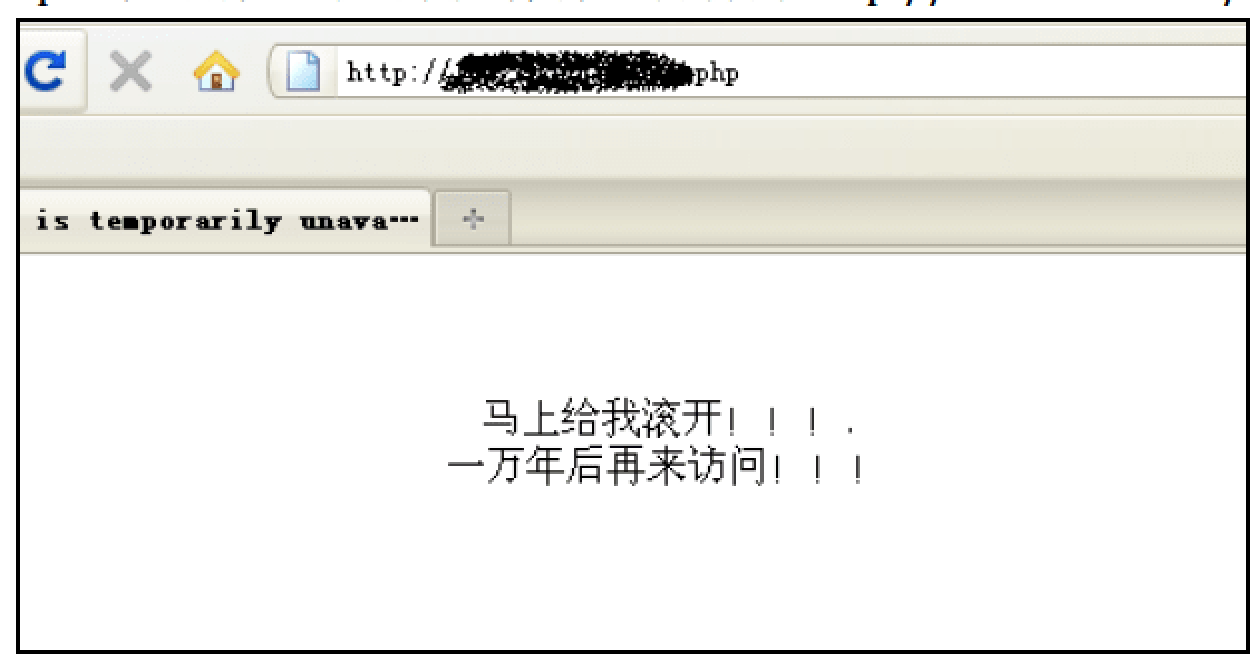
location ~* \.php$ {
 fastcgi_pass 127.0.0.1:9000;
 fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
 fastcgi_param PATH_INFO $fastcgi_script_name;
```

```
 include fastcgi_params;
   }

error_page   500 502 503 504  /gk.html;
location = /50x.html {
root   html;
}
```

最后四行是修改后添加的。

然后（如果 php-fpm 在运行，那么就先停掉）再访问 http://192.168.3.25/test.php：



这样就符合你的需求了。

第二种方法：修改源码重新编译安装 Nginx。

在源码中找到下面的文件，然后做适当的修改：

```
[root@mail src]# pwd
/root/nginx-0.8.53/src
[root@mail src]# vi http/ngx http special response.c

...

static char ngx_http_error_502_page[] =
"<html>" CRLF
"<head><title>GOOD！！！</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1> GOOD！！！</h1></center>" CRLF
;

...
```

在这里我们将两处“**502 Bad Gateway**”替换为“**GOOD！！！**”，然后重新编译，安装一个新的 Nginx 服务器，使用原来的配置文件，但是要将第一种方法中提到的“**error_page**”命令注释掉：

```
    location ~* \.php$ {
  fastcgi_pass 127.0.0.1:9000;
  fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
  fastcgi param PATH INFO $fastcgi script name;
  include fastcgi params;
    }


 #error_page   500 502 503 504  /gk.html;
 #location = /50x.html {
 #root   html;
 #}
```

停掉后端服务器，然后访问：



这样就不是 502 错误了，很好地掩盖了你的责任！

6.3 504 问题—— 504 gateway time-out

第一种方法：如同上面“502 问题””的解决方法。

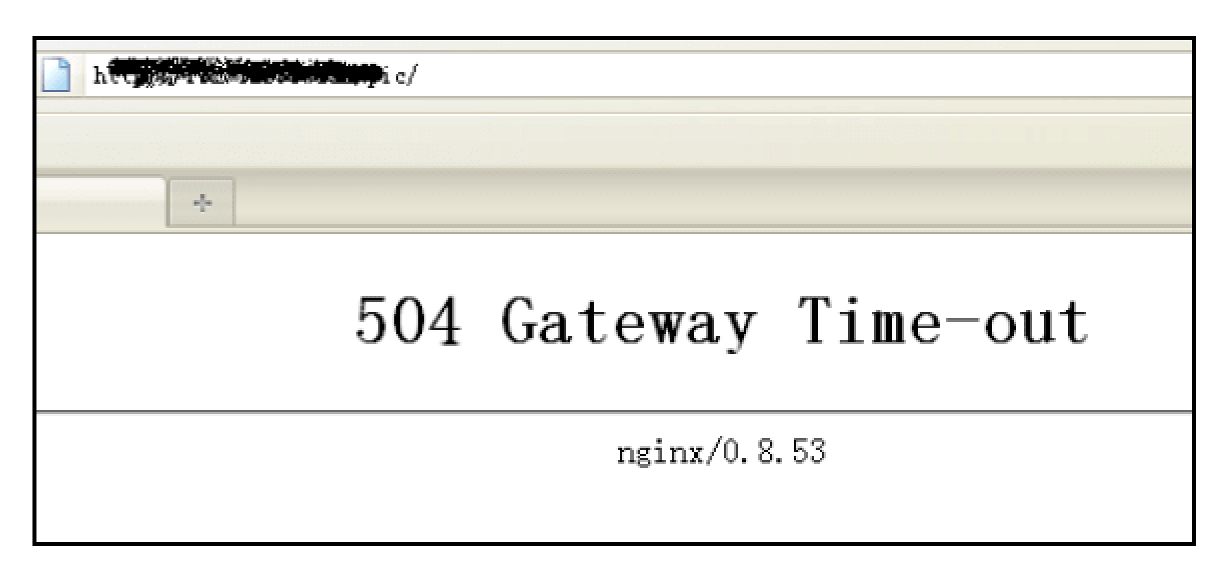
第二种方法：解决方法和上面的一样，具体如下：

先查看 Nginx 配置：

```
  location /pic/ {
 ssi off ;
 proxy pass  http://192.168.9.19/;

   }
```

然后停掉 192.168.9.19 的相关服务，再访问：



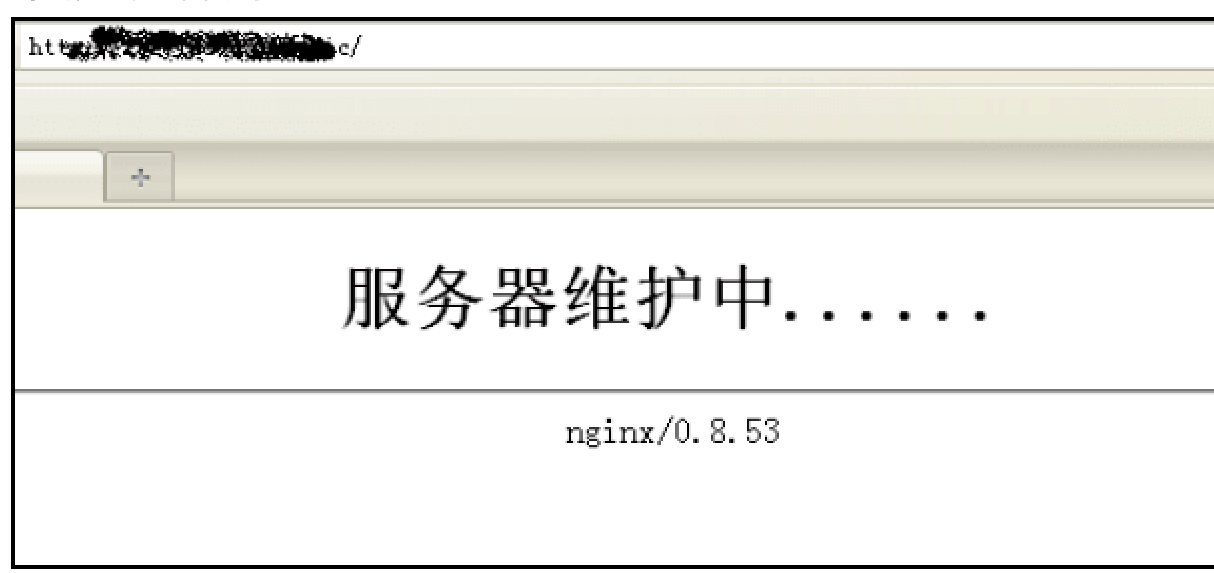
修改源代码 src/http/ngx_http_special_response.c，找到如下部分：

```
static char ngx_http_error_504_page[] =
"<html>" CRLF
"<head><title>504 Gateway Time-out</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>504 Gateway Time-out</h1></center>" CRLF
;
```

修改为以下内容：

```
…
static char ngx_http_error_504_page[] =
"<html>" CRLF
"<head><title>服务器维护中......</title></head>" CRLF
"<body bgcolor=\"white\">" CRLF
"<center><h1>服务器维护中......</h1></center>" CRLF
;
…
```

重新编译 Nginx，然后再访问：



故障虽然是隐藏了，可只能骗得了别人一时，最终还得解决问题。无论是 502 错误还是 504 错误，都有可能是 Nginx 的相关错误，也有可能是后端服务器的问题。因此我们以 php-fpm 为例来分析一下问题的所在。我们在一开始就说了，由于后端服务器发生故障，php-cgi 进程数不够用、php 执行时间长或者是 php-cgi 进程死掉，以及 Nginx 端 FastCGI 缓存使用情况（如果你使用的代理，那么要注意代理缓存的使用情况），那么我们就要从这些问题入手来了解一下问题的所在。

（1）首先需要确定的是后端服务器启动没有，当然在这里就是 php-fpm 进程启动没有。如果该进程没有启动，或者是因为某种原因出错而退出，那么得到的访问结果肯定是“502 Bad Gateway”：

```
[root@mail php5.3.4]# ps -ef|grep php-fpm
root  15375      1    0  12:46  ?      00:00:00  php-fpm:  master  process
   (/usr/local/php5.3.4/etc/php-fpm.conf)
nobody   15376 15375  0 12:46 ?00:00:00 php-fpm: pool www
…//省略
nobody   15380 15375  0 12:46 ?00:00:00 php-fpm: pool www
nobody   15381 15375  0 12:46 ?00:00:00 php-fpm: pool w3
nobody   15382 15375  0 12:46 ?00:00:00 php-fpm: pool w3
…//省略
nobody   15385 15375  0 12:46 ?00:00:00 php-fpm: pool w3
```

（2）其次是确定 php-fpm 的 worker 进程是否够用，如果不够用，对于访问来说就如同后台服务没有开启一样：

计算开启的 worker 进程数目：

```
[root@mail php5.3.4]# ps -ef |grep php-fpm |wc -l
122
```

然后再将这个减去 2（一个是主进程 master，另一个是这里的 grep）就是你开启的 php-fpm 的 worker 进程。当然，如果不嫌麻烦的话，可以使用以下的命令：

```
[root@mail  php5.3.4]#  ps  -ef  |grep  php-fpm  |grep  -v  "master"|grep  -v
    "grep"|wc -l
120
```

计算被使用的 worker 进程（正在处理请求的进程）：

```
[root@mail php5.3.4]# netstat -anp|grep php-fpm|wc -l
124
```

然后再把这个值至少减去 2 或是更多 （一个是监听 LISTEN，另一个是 php-fpm.conf）。LISTEN 的个数要看使用进程池的数量，例如，在这里我们开启的是两个进程池 www 和 w3。如果不怕麻烦，那么执行以下命令更好：

```
[root@mail php5.3.4]# netstat -anp | grep "php-fpm"|grep -v "LISTEN"|grep
    -v "php-fpm.conf"|wc -l
120
```

如果这两个值相近，那么可以考虑增加 worker 进程的数量。

（3）FastCGI 缓冲（buffer）或是代理的缓存情况。在 FastCGI 模块中，与缓存有关的指令有以下两条：

```
fastcgi_buffer_size  4k
fastcgi_buffers   16 4k
```

因为第一条指令的设置依赖于操作系统对内存页面的设置，所以它可以从操作系统查出来：

```
[root@s8 ~]# getconf PAGESIZE
4096
```

这个数值单位是字节（byte），也就是说 4KB。

第二条指令的参数指定将使用多大的缓存区来读取从 FastCGI 进程到来的应答头，这个值由这两条命令的结果 16*4KB=64KB 决定。这意味着所有 FastCGI 返回的应答，Nginx 将超过 64KB

的部分写入磁盘，而 64KB 以内的部分写入内存。如果你设置的等待时间太短，机器又繁忙，势必会造成 502 问题。

如果使用的是代理模块，那么会是以下的设置：

```
proxy_buffer_size  16k;
proxy_buffers  4 16k;
```

对于解释方法等同于 FastCGI 缓存。

（4）php 执行时间长。在 FastCGI 模块中，与时间设置有关的指令有三条，对于这个问题不便举例，这里举一个比较极端的例子，假如你将 fastcgi_send_timeout 的值设置为 1 秒，而实际需要 5 秒才能完成操作，那么这种情况下肯定就是 502 问题了。因此，你需要根据实际情况来调整以下三条指令的值：

```
fastcgi_connect_timeout 60;
fastcgi send timeout   60;
fastcgi_read_timeout   60;
```

不同网站应用的内容不同，要求也不同，因此只能根据应用情况适当地修改这三条配置指令。

如果是使用代理模块，那么相关的设置如下：

```
proxy_connect_timeout 60;
proxy_send_timeout60;
proxy read timeout60;

fail timeout  30;
```

以上命令的意思和 FastCGI 模块的相似。

其中最后一条是 upstream 模块的指令，如果使用了负载均衡，那么参考该指令的使用设置。

> 最后，作为一种学习和参考的方法，要经常去查看 php-fpm 的日志，特别是在出问题的时候，要将日志的级别调整为“DEBUG”级别，以便了解问题的出处。有关日志的部分，请参考本书的相关章节。

第 7 章　使用 TCMalloc 优化 Nginx

TCMalloc 即 Thread-Caching Malloc 的缩写，它是由 Google 公司开发的一款开源工具 google-perftools 中的一成员。TCMalloc 在内存的分配上效率和速度要比标准的 glibc 库好得多，它不但可以用来优化高并发下的 MySQL，从而降低系统的负载，还可以用于 Nginx 实现同样的功能。因此，对于高并发的 Nginx 来说无疑是如虎添翼。

TCMalloc 的优势体现在：

（1）分配内存页的时候，直接跟 OS 打交道，而常用的内存池一般是基于别的内存管理器上分配，如果完全一样的内存管理策略，很明显，TCMalloc 在性能及内存利用率上省掉了第三方内存管理的开销。之所以会出现这种情况，是因为大部分写内存池的 coder 都不太了解 OS；

（2）大部分的内存池只负责分配，不管回收。当然了，没有回收策略，也有别的方法解决问题。比如线程之间协调资源，模块索引一般是一写多读，也就是只有一个线程申请和释放内存，所以不存在线程之间的资源协调；为了避免某些块大量空闲，常用的做法是减少内存块的种类，提高复用率，这样可能会造成内部碎片较多，如果空闲的内存实在太多，可以直接重启。

作为一个通用的内存管理库，TCMalloc 也未必能超过专用的比较粗糙的内存池。比如应用中主要用到的 7 种长度的块，专用的内存池可以只分配这 7 种长度，使得内部没有碎片。或者利用统计信息设置内存池的长度，也可以使得内部碎片比较少，以前本人做过一个工作是统计历史上的需求，然后用动态规则去算内存池长度设置，可以使得内部碎片很少，长度分布发生改变，则重启。

所以 TCMalloc 的意义在于，不需要增加任何开发代价，就能使得内存的开销比较少，而且可以从理论上证明，最优的分配不会比 TCMalloc 的分配好很多。

—— 来源于互联网

7.1 相关安装

根据 32 位或者是 64 位的需求，需要安装不同的软件和安装的配置。

1. 安装 google-perftools

```
[root@mail ~]# wget http://google-perftools.googlecode.com  \
> /files/google-perftools-1.8.3.tar.gz
[root@mail ~]# tar -zxvf google-perftools-1.8.3.tar.gz
[root@mail ~]# cd google-perftools-1.8.3
[root@mail google-perftools-1.8.3]# ./configure
[root@mail google-perftools-1.8.3]#make
[root@mail google-perftools-1.8.3]#make install
```

两点注意事项

（1）如果安装 google-perftools 时没有安装在标准的位置，在安装 Nginx 时会使用这些库文件，那么将会出现无法找到的问题。通常的做法是使用以下方法将库文件载入：

```
[root@mail lib]# vi /etc/ld.so.conf
/usr/local/google-perftools/lib

[root@mail lib]# ldconfig
```

但是你会发现在安装 Nginx 时仍然会出现以下错误：

```
checking for Google perftools ... not found
checking for Google perftools in /usr/local/ ... not found
./configure: error: the Google perftool module requires the Google perftools
library. You can either do not enable the module or install the library.
```

这说明 Nginx 在安装时只是查找了/usr/local 目录。如果是因为这样的问题导致 Nginx 无法安装，可以把 lib/下的库文件全部复制到/usr/local/lib 目录。例如，在安装 google-perftools 时，使用了“--prefix=/usr/local/google-perftools”，它的目录结构为：

```
[root@mail local]# tree -L 1 google-perftools
google-perftools
|-- bin
|-- include
|-- lib
'-- share

4 directories, 0 files
```

（2）参照 google-perftools 安装包中的 INSTALL 文件，如果所在的 Linux 系统为 64 位系统，那么首先需要安装 libunwind 软件：

```
[root@mail ~]# wget http://download.savannah.gnu.org/   \
> releases/libunwind/libunwind-1.0.tar.gz
[root@mail ~]#tar zxvf libunwind-1.0.tar.gz
[root@mail libunwind-1.0]#cd libunwind-1.0/
[root@mail libunwind-1.0]#CFLAGS=-fPIC ./configure
[root@mail libunwind-1.0]#make CFLAGS=-fPIC
[root@mail libunwind-1.0]#make CFLAGS=-fPIC install
```

在安装 google-perftools 时还需要添加-- enable-frame-pointers 选项。

2. 安装 Nginx

```
[root@mail nginx-1.0.2]# ./configure --prefix=/usr/local/nginx-1.0.2
        -google  \
> --with-google perftools module
[root@mail nginx-1.0.2]make
[root@mail nginx-1.0.2]make install
```

该模块用于为 worker 启用 Google 性能分析工具（Google Performance Tools），它在 0.6.29

以上的 Nginx 版本中可用。由于在 Nginx 的默认安装中没有选择该模块，因此，要想使用该模块，那么就在编译安装时添加--with-google_perftools_module 选项。

7.2 配置示例

```
google_perftools_profiles /path/to/profile;
```

7.3 指令

指令名称：google_perftools_profiles

语法：google_perftools_profiles path

默认值：none

功能：该指令指定了 profile 的基本文件名，worker 的 PID 将会附加在该文件名称之后。

7.4 使用实例

在配置文件中添加以下内容。要注意 google_perftools_profiles 在配置文件中的位置。

```
[root@mail conf]# more nginx.conf
user  nobody;
worker processes  1;

events {
worker connections  1024;
}

google_perftools_profiles /tmp/tcmalloc;

http {
include   mime.types;
default_type  application/octet-stream;

sendfileon;
keepalive_timeout  65;

server {
listen   80;
server_name  localhost;
```

```
location / {
root   html;
index  index.html index.htm;
}
}
}
```

启动 Nginx 后，执行以下命令查看：

```
[root@mail conf]# lsof -n | grep tcmalloc
nginx  11746 nobody 10w  REG 253,0  0 11850490  /tmp/tcmalloc.11746
```

这个结果表示 TCMalloc 生效。

第 8 章　PCRE 正则表达式

由于在使用 Nginx 的过程中会用到正则表达式，因此有必要对这一部分内容进行一个清晰的说明。

> 为了在 Nginx 中匹配中文正则表达式，在安装时要选择--enable-utf8 和--enable-unicode-properties。

8.1 安装 PCRE

PCRE 的安装比较简单，很规矩的三步：configure、make 和 make install。

当执行完成 configure 后一定要查看一下：

```
[root@mail pcre-8.13]# ./configure --prefix=/usr/local/pcre-8.13 /
 --enable-utf8 --enable-unicode-properties


pcre-8.13 configuration summary:

Install prefix .................. : /usr/local/pcre-8.13
C preprocessor .................. : gcc -E
C compiler ...................... : gcc
C++ preprocessor ................ : g++ -E
C++ compiler .................... : g++
Linker .......................... : /usr/bin/ld
C preprocessor flags ............ :
C compiler flags ................ : -O2
C++ compiler flags .............. : -O2
Linker flags .................... :
Extra libraries ................. :

Build C++ library ............... : yes
Enable UTF-8 support ............ : yes
Unicode properties .............. : yes
Newline char/sequence ........... : lf
\R matches only ANYCRLF ......... : no
EBCDIC coding ................... : no
Rebuild char tables ............. : no
Use stack recursion ............. : yes
POSIX mem threshold ............. : 10
```

```
Internal link size .............. : 2
Match limit ..................... : 10000000
Match limit recursion ........... : MATCH_LIMIT
Build shared libs ............... : yes
Build static libs ............... : yes
Buffer size for pcregrep ........ : 20480
Link pcregrep with libz ......... : no
Link pcregrep with libbz2 ....... : no
Link pcretest with libreadline .. : no
```

编译：

```
[root@mail pcre-8.13]#make
```

安装：

```
[root@mail pcre-8.13]#make install

…

----------------------------------------------------------------------
Libraries have been installed in:
   /usr/local/pcre-8.13/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the `-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD LIBRARY PATH' environment variable
 during execution
   - add LIBDIR to the 'LD RUN PATH' environment variable
 during linking
   - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------

…
```

下面是安装完成后的目录结构：

```
[root@mail local]# tree pcre-8.13/
pcre8/
|-- bin
|   |-- pcre-config
```

```
|   |-- pcregrep
|   '-- pcretest
|-- include
…
|   |-- pcrecpparg.h
|   '-- pcreposix.h
|-- lib
|   |-- libpcre.a
…
|   |-- dibpcreposix.so.0.0.0
|   '-- pkgconfig
|   |-- libpcre.pc
|   |-- libpcrecpp.pc
|   '-- libpcreposix.pc
'-- share
|-- doc
|   '-- pcre
…
|   |-- html
…
|   |-- pcregrep.txt
|   '-- pcretest.txt
'-- man
|-- man1
|   |-- pcre-config.1
|   |-- pcregrep.1
|   '-- pcretest.1
'-- man3
|-- pcre.3
…
|-- pcre_copy_substring.3
…
'-- pcresyntax.3
```

在这个目录结构中：

- bin/：目录中相关的命令；
- include/：目录中是头文件；
- lib/：目录中是库文件；
- share/：目录中是文档信息，包括 doc/和 man/，是 man 文档。

需要注意：一是 bin 目录，它下面有三个命令，二是 man 目录，认识命令的必经之路。

8.2 命令

bin 目录下的三个命令。

- pcretest：用于正则表达式测试；
- pcre-config：用于返回 PCRE 的配置；
- pcregrep：一个具有 PCRE 功能的 grep 工具。

在这里我们要说明的是 pcretest 和 pcre-config 命令。

8.2.1 pcre-config 命令

下面看一下 pcre-config 命令的用法：

```
[root@mail g]# pcre-config -h
Usage:  pcre-config  [--prefix]  [--exec-prefix]  [--version]  [--libs]
  [--libs-posix] [--cflags] [--cflags-posix]
```

使用方法不言而喻，特别是在安装 Nginx 出错时，错误的原因在于找不到 PCRE 时很有用，例如：

```
[root@mail g]# pcre-config --prefix
/usr/local
```

8.2.2 pcretest 命令

pcretest 是一个 PCRE 正则表达式测试程序，你可以用它来检测你所编写的正则表达式是否是你表达的那个意思。

如果 pcretest 有两个文件名参数，那么它会从第一个文件读出，然后再写入第二个文件；如果只指定了一个文件名，那么它会从第一个文件读出，然后再向标准输出设备 stdout 写入；否则，它会从标准输入设备 stdin 读取文件，然后再向标准输出设备 stdout 写入，在这种情况下，当你输入 pcretest 并回车后就会出现一个“re>”命令提示符，这时就可以输入正则表达式了，输入正则表达式后再次回车就出现了另一个提示符“data>”，这时便可以输入验证的字符串了。

1. **用法格式**

```
[root@mail g]# pcretest -help
Usage: pcretest [options] [<input file> [<output file>]]
```

例如：

```
[root@mail bin]# cat a
/^ (.+) ?\.xx\.com$/
[root@mail bin]# ./pcretest -b a d
[root@mail bin]# more  a d
::::::::::::::
a
::::::::::::::
/^ (.+) ?\.xx\.com$/
```

```
::::::::::::::
d
::::::::::::::
PCRE version 8.13 2011-08-16

/^(.+)?\.xx\.com$/
------------------------------------------------------------------
  0  30 Bra
  3 ^
  4 Brazero
  5   7 CBra 1
 10 Any+
 12   7 Ket
 15 .xx.com
 29 $
 30  30 Ket
 33 End
------------------------------------------------------------------
Capturing subpattern count = 1
Options: anchored
No first char
Need char = 'm'
```

或者是以下方式：

```
[root@mail bin]# ./pcretest
PCRE version 8.13 2011-08-16

  re> /^(.+)?\.xx\.com$/
data> www.xx.com
 0: www.xx.com
 1: www
data>
```

2. 选项

该命令提供了以下选项。

选项名称：-b

功能：显示编译代码（bytecode）。

例如：

```
[root@mail bin]# ./pcretest -b
PCRE version 8.13 2011-08-16

  re> /^(.+)?\.xx\.com$/
------------------------------------------------------------------
  0  30 Bra
```

```
  3 ^
  4 Brazero
  5   7 CBra 1
 10 Any+
 12   7 Ket
 15 .xx.com
 29 $
 30  30 Ket
 33 End
------------------------------------------------------------------
data>
```

选项名称：-C

功能：显示 PCRE 在安装时指定的选项并且退出。

例如：

```
[root@mail bin]# ./pcretest -C
PCRE version 8.13 2011-08-16
Compiled with
  UTF-8 support
  Unicode properties support
  Newline sequence is LF
  \R matches all Unicode newlines
  Internal link size = 2
  POSIX malloc threshold = 10
  Default match limit = 10000000
  Default recursion depth limit = 10000000
  Match recursion uses stack
```

选项名称：-d

功能：调试信息，显示编译代码和信息，相当于-b 和-i。

例如：

```
[root@mail bin]# ./pcretest -d
PCRE version 8.13 2011-08-16

  re> /^(.+)?\.xx\.com$/
------------------------------------------------------------------
  0  30 Bra
  3 ^
  4 Brazero
  5   7 CBra 1
 10 Any+
 12   7 Ket
 15 .xx.com
 29 $
```

```
 30  30 Ket
 33 End
------------------------------------------------------------------
Capturing subpattern count = 1
Options: anchored
No first char
Need char = 'm'
data>
```

选项名称：-dfa

功能：该选项用于对强制 DFA 匹配。

选项名称：-help

功能：显示该命令的帮助信息。

选项名称：-i

功能：显示有关被编译模式的信息。

例如：

```
[root@mail bin]# ./pcretest -i
PCRE version 8.13 2011-08-16

  re> /^\d?\d (jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec) \d\d$/
Capturing subpattern count = 1
Options: anchored
No first char
No need char
data> 23ja\P\D
Partial match: 23ja
```

选项名称：-m

功能：该选项用于输出使用内存的信息。

例如：

```
[root@mail bin]# ./pcretest -m
PCRE version 8.13 2011-08-16

  re> /^\d?\d (jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec) \d\d$/
Memory allocation (code space) : 127
data> 23ja\P\D
Partial match: 23j
```

选项名称：-o <n>

功能：该选项用于设置输出矢量的偏移量，默认值为 45。

选项名称：-p

功能：使用 POSIX 接口。

例如：

```
[root@mail bin]# ./pcretest -p
PCRE version 8.13 2011-08-16

  re>  /^\d?\d (jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec) \d\d$/
data> 23ja\P\D
** Can't use dfa matching in POSIX mode: \D ignored
No match: POSIX code 17: match failed
```

选项名称：-q

功能：如果使用该选项，那么则不会输出 PCRE 的版本号。

例如：

```
[root@mail bin]# ./pcretest -p -q
  re>
```

选项名称：-S <n>

功能：该选项用于设置使用堆栈的大小，单位为兆字节。

选项名称：-s

功能：强制每一个模式被学习。

选项名称：-t

功能：在编译时执行。

例如：

```
[root@mail bin]# ./pcretest -t
PCRE version 8.13 2011-08-16

  re> / (tang|tangerine|tan) /
Compile time 0.0023 milliseconds
```

选项名称：-t <n>

功能：在编译时执行，并且执行<n>次。

选项名称：-tm

功能：显示的仅是执行匹配的时间。

例如：

```
 [root@mail bin]# ./pcretest -tm
PCRE version 8.13 2011-08-16

  re> / (tang|tangerine|tan) /
data> yellow tangerine\D
Execute time 0.0010 milliseconds
 0: tangerine
 1: tang
 2: tan
data> yellow sdfsd
Execute time 0.0004 milliseconds
```

```
No match
```

选项名称：-tm <n>

功能：显示执行匹配的时间，重复执行<n>次。

3. 使用 pcretest 命令

正如其名字，该命令是一个 Perl 兼容的正则表达式测试程序。在这里我们举例说明如何验证我们的正则表达式。

例 1：

```
[root@mail bin]# ./pcretest
PCRE version 8.01 2010-01-19

  re> /^(.+)?\.t1\.com$/ //这里是要测试的正则表达式，放置在一对“//”中
data> www.t1.com            //测试字符串 www.t1.com
 0: www.t1.com
 1: www                     //变量 1 的值
data> mail.t1.com           //测试字符串 mail.t1.com
 0: mail.t1.com
 1: mail                    //变量 1 的值
data>
```

例 2：

```
[root@mail bin]# ./pcretest
PCRE version 8.01 2010-01-19

  re> /^(www\.)?(.+)$/ //这里是要测试的正则表达式，放置在一对“//”中
data> www.t1.com           //测试字符串 www.t1.com
 0: www.t1.com
 1: www.                   //变量 1 的值
 2: t1.com                //变量 2 的值
data> www.baidu.com
 0: www.baidu.com
 1: www.                  //变量 1 的值
 2: baidu.com             //变量 2 的值
data>
```

例 3：

```
[root@mail bin]# ./pcretest
PCRE version 8.01 2010-01-19

  re> /\/im\/([0-9]{2})([0-9]{2})([0-9]{2}) //要测试的正则表达式
data> \/im\/123456      //测试字符串/im/123456
 0: /im/123456
 1: 12                 //变量 1 的值
 2: 34                 //变量 2 的值
```

```
 3: 56                  //变量 3 的值
data>
```

例 4：

```
[root@mail bin]# ./pcretest
PCRE version 8.13 2011-08-16

  re> /^(.+)?\.xx\.cn$/8
data> 中国.xx.cn
 0: \x{4e2d}\x{56fd}.xx.cn
 1: \x{4e2d}\x{56fd}
data> 功能.xx.cn
 0: \x{529f}\x{80fd}.xx.cn
 1: \x{529f}\x{80fd}
data>
```

总结

从这 4 个例子中得到以下结论：

- 第 0 个变量所取的值永远是被测试的字符串，可惜这个$0 在 Nginx 中是不能使用的。在 Nginx 中只能从$1 开始使用；
- 关于$1、$2、$3...变量的选择，按照测试表达式中圆括号“（）”的顺序，最先出现的为 1，然后依次类推。

对变量的应用

在前面的 3 个例子中，对于例 1 和例 2 的应用我们再清楚不过了，而对于例 3，应用如下：

```
rewrite  "/im/([0-9]{2})([0-9]{2})([0-9]{2})" /sdc/im/$1/$1$2/
  $1$2$3.jpg;
```

因此，对于一个“/im/123456”请求，将会重写为/im/12/34/56/123456.jpg 格式。

另外，不同的 PCRE 版本对正则表达式的解释可能也有差异，因此要注意先测试再使用。

8.3 man 目录

我们要了解的第二个目录是 man 目录，这个目录是一个 man 文档目录，如果我们在安装 PCRE 时没有安装在标准目录（系统预定的目录）下，那么使用 man 命令就不能找到命令的帮助信息。在这种情况下，可以在“/etc/man.config”中找到“MANPATH”部分，然后添加 man 文档所在的路径：

```
MANPATH /usr/local/pcre8/man
```

8.4 Nginx 与正则表达式

由于在具体使用 Nginx 的过程中需要正则表达式的地方很多，因此我们在安装 Nginx 的时候

要选择这个功能。

> 提示：有关安装 PCRE 在 8.1 小节中讲过，但是由于我们要使用到 UTF-8，因此有必要再在这里介绍一下。

8.4.1 正则表达式支持 UTF-8

在启用安装 PCRE 时我们指定了“--enable-utf8 --enable-unicode-properties”，但是我们在安装 Nginx 时使用了“--with-pcre=/usr/local/pcre-8.13/lib/”，并没像我们所想的那样安装，而是出错了：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8-pcre  \
> --with-pcre=/usr/local/pcre-8.13/lib/


Configuration summary
  + using PCRE library: /usr/local/pcre-8.13/lib/
  + OpenSSL library is not used
  + md5: using system crypto library
  + sha1: using system crypto library
  + using system zlib library

[root@mail nginx-1.0.8]# make
make -f objs/Makefile
make[1]: Entering directory '/root/nginx-1.0.8'
cd /usr/local/pcre-8.13/lib/ \
&& if [ -f Makefile ]; then make distclean; fi \
&& CC="gcc" CFLAGS="-O2 -fomit-frame-pointer -pipe " \
./configure --disable-shared
/bin/sh: line 2: ./configure: No such file or directory
make[1]: *** [/usr/local/pcre-8.13/lib//Makefile] Error 127
make[1]: Leaving directory '/root/nginx-1.0.8'
make: *** [build] Error 2
```

因此，我们需要使用指定 PCRE 的源码“--with-pcre=/root/pcre-8.13”，应该按如下所示的方式安装 Nginx：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8-pcre \
> --with-pcre=/root/pcre-8.13
```

但是在执行 make 命令时会发现：

```

pcre-8.13 configuration summary:

Install prefix .................. : /usr/local
C preprocessor .................. : gcc -E
```

```
C compiler ...................... : gcc
C++ preprocessor ................ : g++ -E
C++ compiler .................... : g++
Linker .......................... : /usr/bin/ld
C preprocessor flags ............ :
C compiler flags ................ : -O2 -fomit-frame-pointer -pipe
C++ compiler flags .............. : -O2
Linker flags .................... :
Extra libraries ................. :

Build C++ library ............... : yes
Enable UTF-8 support ............ : no
Unicode properties .............. : no
Newline char/sequence ........... : lf

…
```

这是因为我们在编译时使用了 PCRE 的源码，而在默认安装 PCRE 时是不支持 utf8 的，因此有必要添加这个编译选项，在执行完成./configure 命令之后：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8-pcre  \
> --with-pcre=/root/pcre-8.13
```

编辑以下文件，这个目录是在执行./configure 命令之后生成的，找到以下内容：

```
vi ./objs/Makefile

…

  1054 /root/pcre-8.13/pcre.h: /root/pcre-8.13/Makefile
  1055
  1056 /root/pcre-8.13/Makefile:   objs/Makefile
  1057 cd /root/pcre-8.13 \
  1058 && if [ -f Makefile ]; then $(MAKE) distclean; fi \
  1059 && CC="$(CC)" CFLAGS="-O2 -fomit-frame-pointer -pipe " \
  1060 ./configure --disable-shared
  1061

…
```

如果没有添加额外的模块，那么大致就在 1054～1060 行，显然这里是对 PCRE 的编译选项进行设置。我们可以在这里添加“--enable-utf8 --enable-unicode-properties”两个选项，例如：

```
vi ./objs/Makefile

…
```

```
 /root/pcre-8.13/pcre.h: /root/pcre-8.13/Makefile

 /root/pcre-8.13/Makefile:   objs/Makefile
cd /root/pcre-8.13 \
&& if [ -f Makefile ]; then $ (MAKE) distclean; fi \
&& CC="$ (CC) " CFLAGS="-O2 -fomit-frame-pointer -pipe " \
./configure --disable-shared --enable-utf8 --enable-unicode-properties


…
```

然后再次执行 make：

```
pcre-8.13 configuration summary:

Install prefix .................. : /usr/local
C preprocessor .................. : gcc -E
C compiler ...................... : gcc
C++ preprocessor ................ : g++ -E
C++ compiler .................... : g++
Linker .......................... : /usr/bin/ld
C preprocessor flags ............ :
C compiler flags ................ : -O2 -fomit-frame-pointer -pipe
C++ compiler flags .............. : -O2
Linker flags .................... :
Extra libraries ................. :

Build C++ library ............... : yes
Enable UTF-8 support ............ : yes
Unicode properties .............. : yes

…
```

可以了，这次支持 UTF8 了。

8.4.2 Nginx 使用正则表达式

在使用 Nginx 中，有很多地方要使用到 pcre 正则表达式，例如 if 中使用过正则表达式、rewrite 模块中使用过正则表达式、server_name 中同样使用了正则表达式，等等。这些匹配参考相关各个章节就可以了，这里不再重复介绍了。

下面重点要讲述的一点是关于中文使用正则表达式。

```
location /{
  rewrite " (*UTF8) ^/ ([\x{4e00}-\x{9fbf}]+) " /$1.html break;
```

```
# rewrite "(*UTF8)^/([\x{4e00}-\x{9fbf}]+)/([\x{4e00}-\x{9fbf}]+)"
  /$1/$2.html break;
# rewrite "(*UTF8)^/(..)$" /$1.html break;
# rewrite ^/(......)$ /$1.html break;
}
```

以下是我们的访问目录结构：

```
[root@mail html]# tree
.
|-- 2.html
|-- 50x.html
|-- index.html
|-- ll
|   '-- index.html
|-- xx
|   '-- 文章.html
|-- 作用.html
|-- 你好.html
|-- 功能
|   |-- 2.html
|   |-- index.html
|   '-- 功能.html
'-- 我不知道.html

3 directories, 11 files
```

在上面的配置中，要想访问到这个目录结构中的“我不知道.html”、“作用.html”、“2.html”和“你好.html”文件，则需要使用第一条配置，它能够使用与所有中文文件名称的匹配。

如果要访问“功能/2.html”、“功能/功能.html”，那么需要使用第二条配置。

第三条和第四条配置功能相同，它们都用于匹配两个中文字符的文件名称，区别在于，第三条配置使用了（*UTF8），在正则表达式中使用了两个点号，而第四条配置没有使用（*UTF8），却使用了 6 个点号，如果要匹配三个字符的中文文件名称，那么则需要 9 个点号，例如：

```
rewrite ^/(.........)$ /$1.html break;
```

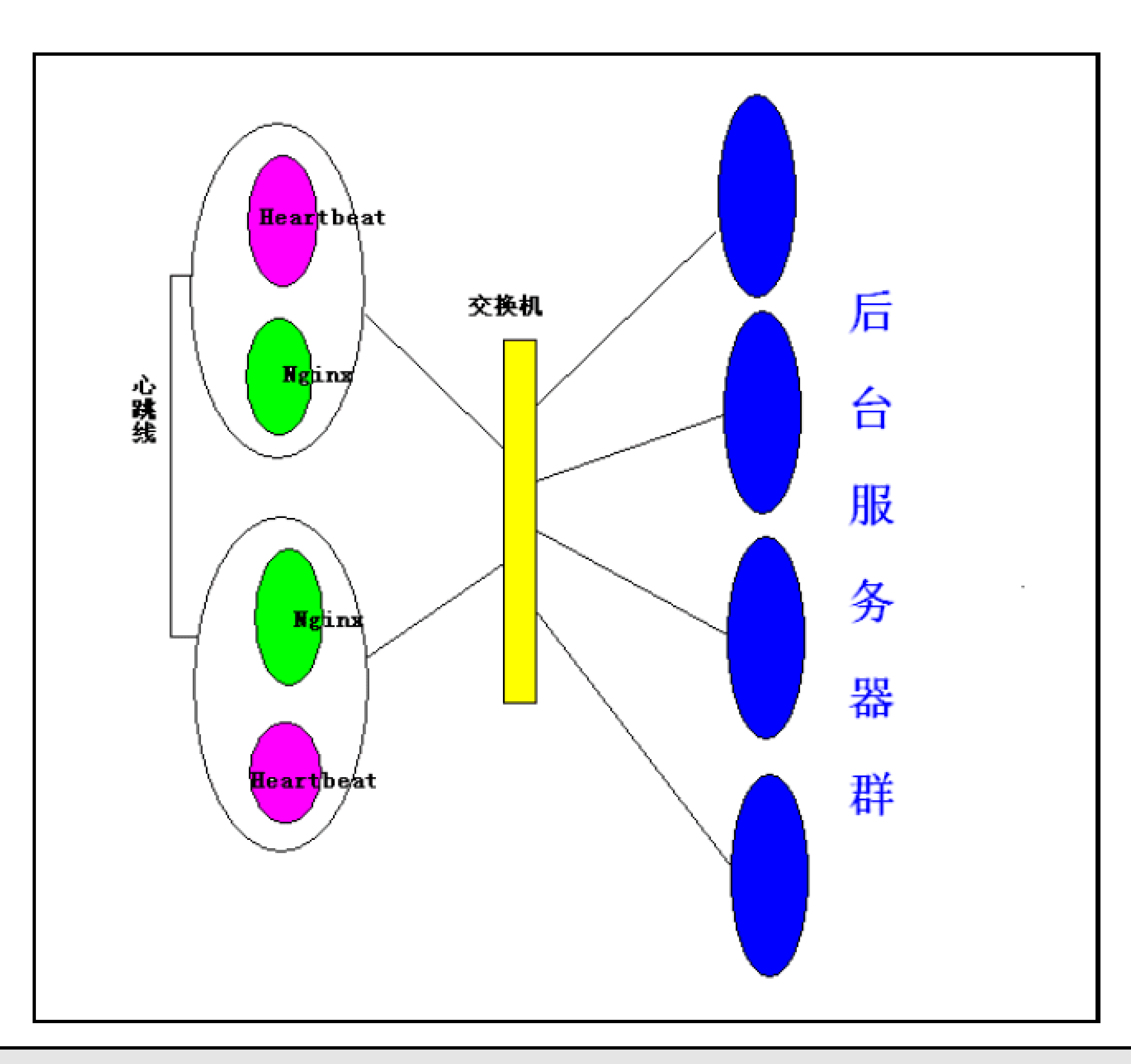
而要使用（*UTF8），则有 3 个点号就可以了，例如：

```
rewrite "(*UTF8)^/(...)$" /$1.html break;
```

第 9 章　Nginx 高可用的实现

在 Nginx 下实现高可用不是担心 Nginx 服务出问题，而是担心硬件的问题。因此在设计通过 Heartbeat 服务来提供高可用时没有通过 Heartbeat 服务器来控制 Nginx 服务器的启动、停止等，而只是通过它来提供一个“浮动”的 IP 地址，此 IP 就是 Nginx 服务器提供访问的 IP，也就是被解析到相应域名的 IP 地址。

示意图：



> 这里需要注意的一点是，作为高可用，一般的交换机都是单电源，因此无须考虑交换机单电源的故障点，这不是我们讨论的范畴，只是提醒一下。

9.1 安装 Heartbeat

3.0 以后的 Heartbeat 由 glue、heartbeat 和 agents 三部分组成。因此，我们需要分别安装。下面是一些安装之前必须安装的工具：

- C 编译器（典型的就是 GCC）和与 C 相关的开发文档；
- flex 和 bison 解析编译；
- net-snmp 开发头文件；
- OpenIPMI 开发头；
- Python 解释器。

添加用户和组：

在安装之前首先要添加 Heartbeat 服务运行的用户和组：

```
[root@slave ~]# groupadd haclient
[root@slave ~]# useradd -g haclient hacluster
```

在 master 和 slave 节点上都要添加运行该服务的用户和组。

系统环境：

```
[root@master ~]# more /etc/redhat-release
Red Hat Enterprise Linux Server release 5.6 (Tikanga)
```

9.1.1 下载安装 glue

下载 glue：

```
[root@master ~]# wget http://hg.linux-ha.org/glue/archive/glue-
  1.0.8.tar.bz2
--09:11:34--  http://hg.linux-ha.org/glue/archive/glue-1.0.8.tar.bz2
Resolving hg.linux-ha.org... 66.35.48.29
Connecting to hg.linux-ha.org|66.35.48.29|:80... connected.
HTTP request sent, awaiting response... 200 Script output follows
Length: unspecified [application/x-tar]
Saving to: 'Reusable-Cluster-Components-glue--glue-1.0.8.tar.bz2'
```

编译安装：

```
[root@master ~]#tar -jxvf Reusable-Cluster-Components-glue--glue-
   1.0.8.tar.bz2
[root@master ~]#cd Reusable-Cluster-Components-glue--glue-1.0.8

[root@master Reusable-Cluster-Components-glue--glue-1.0.8]#./autogen.sh
[root@master Reusable-Cluster-Components-glue--glue-1.0.8]# ./configure
cluster-glue configuration:
  Version  = 1.0.8 (Build: c69dc6ace936f501776df92dab3d611c2405f69e)
  Features =

  Prefix   = /usr
  Executables  = /usr/sbin
  Man pages= /usr/man
  Libraries= /usr/lib
  Header files = /usr/include
  Arch-independent files   = /usr/share
  Documentation= /usr/share/doc
  State information= /usr/var
  System configuration = /usr/etc

  Use system LTDL  = no
```

```
    HA group name= haclient
    HA user name = hacluster

    CFLAGS  = -g -O2 -ggdb3 -O0  -fgnu89-inline -fstack-protector-all -Wall
 -Waggregate-return-Wbad-function-cast-Wcast-qual-Wcast-align-Wdeclaration-
 after-statement-Wendif-labels-Wfloat-equal-Wformat=2-Wformat-security-Wfor
 mat-nonliteral-Winline-Wmissing-prototypes-Wmissing-declarations-Wmissing-
 format-attribute-Wnested-externs-Wno-long-long-Wno-strict-aliasing-Wpointe
 r-arith-Wstrict-prototypes-Wwrite-strings-ansi-D_GNU_SOURCE-DANSI_ONLY-Wer
 ror
    Libraries= -lbz2 -lxml2 -lc -luuid -lrt -ldl  -L/lib -lglib-2.0
    Stack Libraries  =
   [root@master Reusable-Cluster-Components-glue--glue-1.0.8]# make
   [root@master Reusable-Cluster-Components-glue--glue-1.0.8]# make install
```

9.1.2 下载安装 Heartbeat

下载并解压 Heartbeat：

```
   [root@master ~]#wget http://hg.linux-ha.org/heartbeat-STABLE 3 0/archive
/7e3a82377fa8.
   tar.bz2
   --09:51:57--
 http://hg.linux-ha.org/heartbeat-STABLE_3_0/archive/7e3a82377fa8.tar.bz2
   Resolving hg.linux-ha.org... 66.35.48.29
   Connecting to hg.linux-ha.org|66.35.48.29|:80... connected.
   HTTP request sent, awaiting response... 200 Script output follows
   Length: unspecified [application/x-tar]
   Saving to: 'Heartbeat-3-0-7e3a82377fa8.tar.bz2'
   [root@master ~]# tar -jxvf Heartbeat-3-0-7e3a82377fa8.tar.bz2
   [root@master ~]# cd Heartbeat-3-0-7e3a82377fa8
   [root@master Heartbeat-3-0-7e3a82377fa8]# ./bootstrap
   Autoconf package autoconf found.
   Automake package automake-1.9 found.
   Libtool package libtool found.
   aclocal-1.9
   autoheader
   libtoolize --ltdl --force --copy
   aclocal-1.9
   automake-1.9 --add-missing --include-deps --copy
   configure.in: installing './install-sh'
   configure.in: installing './missing'
   buildtools/Makefile.am: installing './depcomp'
   cts/Makefile.am:24: installing './py-compile'
   heartbeat/Makefile.am: installing './compile'
```

```
autoconf
Now run ./configure.
[root@master Heartbeat-3-0-7e3a82377fa8]#
```

执行 configure：

```
[root@master Heartbeat-3-0-7e3a82377fa8]# ./configure
checking build system type... i686-redhat-linux-gnu
…
configure: WARNING: value/default "--localstatedir=/usr/local/var" is
poor.
configure: WARNING: "/var/something" is strongly recommended.
configure: WARNING: We also recommend using "ConfigureMe".
configure: WARNING: Sleeping for 10 seconds.
…
checking for heartbeat/glue_config.h... yes
checking glue_config.h usability... no
checking glue_config.h presence... no
checking for glue config.h... no
configure: error: Core development headers were not found
See 'config.log' for more details.
```

由于出错，于是执行以下配置命令：

```
[root@master Heartbeat-3-0-7e3a82377fa8]# ./ConfigureMe configure

Configure flags for RedHat Linux: --prefix=/usr --sysconfdir=/etc
--localstatedir=/var --mandir=/usr/share/man --disable-rpath
Running ./configure --prefix=/usr --sysconfdir=/etc --localstatedir=/var
--mandir=/usr/share/man --disable-rpath
…
heartbeat configuration:
  Version  = "3.0.5"
  Executables  = "/usr/sbin"
  Man pages= "/usr/share/man"
  Libraries= "/usr/lib"
  Header files = "/usr/include"
  Arch-independent files   = "/usr/share"
  Documentation files  = "/usr/share/doc/"
  State information= "/var"
  System configuration = "/etc"
  Init (rc) scripts= "/etc/init.d"
  Init (rc) defaults   = "/etc/sysconfig"
  Use system LTDL  = "no"
  HA group name= "haclient"
  HA group id  = "501"
```

```
     HA user name = "hacluster"
     HA user user id  = "500"
     Build dopd plugin= "yes"
     Enable times kludge  = "yes"
   CC WARNINGS="-Wall-Wmissing-prototypes-Wmissing-declarations-Wstrict-pro
totypes-Wdeclaration-after-statement-Wpointer-arith-Wwrite-strings-Wcast-qua
l-Wcast-align-Wbad-function-cast-Winline-Wmissing-format-attribute-Wformat=2
-Wformat-security-Wformat-nonliteral-Wno-long-long-Wno-strict-aliasing-Werro
r "
   MangledCFLAGS="-g-O2-I/usr/include/heartbeat-Wall-Wmissing-prototypes-Wm
issing-declarations-Wstrict-prototypes-Wdeclaration-after-statement-Wpointer
-arith -Wwrite-strings -Wcast-qual-Wcast-align -Wbad-function-cast -Winline
-Wmissing-format-attribute-Wformat=2-Wformat-security-Wformat-nonliteral-Wno
-long-long-Wno-strict-aliasing-Werror-ggdb3-funsigned-char"
     Libraries= "-lbz2 -lz -lc -luuid -lrt -ldl "
     RPATH enabled= "no"
     Distro-style RPMs= "no"

   Note: If you use the 'make install' method for installation you
   also need to adjust '/etc/passwd' and '/etc/group' manually.
```

编译并安装：

```
[root@master Heartbeat-3-0-7e3a82377fa8]# make
[root@master Heartbeat-3-0-7e3a82377fa8]# make install
```

9.1.3 安装 agents

下载并安装 agents：

```
   [root@master ~]# wget https://nodeload.github.com/ClusterLabs/resource
-agents/tarball
   /v3.9.2
   --09:35:52--
https://nodeload.github.com/ClusterLabs/resource-agents/tarball/v3.9.2
   Resolving nodeload.github.com... 207.97.227.252
   Connecting to nodeload.github.com|207.97.227.252|:443... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 452498 (442K) [application/octet-stream]
   Saving to: 'ClusterLabs-resource-agents-v3.9.2-0-ge261943.tar.gz'
   [root@master ~]#tar -zxvf ClusterLabs-resource-agents-v3.9.2-0
-ge261943.tar.gz
   [root@master ClusterLabs-resource-agents-b735277]# tar -zxvf ClusterLabs
-resource-agents-v3.9.2-0-ge261943.tar.gz
   [root@slave ClusterLabs-resource-agents-b735277]# ./autogen.sh
```

```
autoreconf: Entering directory '.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal
configure.ac:9: error: Autoconf version 2.63 or higher is required
configure.ac:9: the top level
autom4te: /usr/bin/m4 failed with exit status: 63
aclocal: autom4te failed with exit status: 63
autoreconf: aclocal failed with exit status: 63
```

在上面执行 autogen.sh 脚本时出现 autoconf 版本低的问题，因此有必要安装一个高版本的 autoconf 工具。

安装 autoconf-2.68：

```
[root@master ~]#tar -zxvf autoconf-2.68.tar.gz
[root@master ~]#cd autoconf-2.68
[root@master autoconf-2.68]#make
[root@master autoconf-2.68]#make install
```

再次执行 autogen.sh：

```
autoconf/lang.m4:198: AC LANG CONFTEST is expanded from...
autoconf/general.m4:2599: _AC_COMPILE_IFELSE is expanded from...
autoconf/general.m4:2609: AC_COMPILE_IFELSE is expanded from...
configure.ac:713: the top level
configure.ac:63: require Automake 1.10.1, but have 1.9.6
autoreconf: automake failed with exit status: 1
```

再次执行 autogen.sh 时发现 automake 的版本低，因此也必须安装一个高版本的 automake 工具。

安装 automake：

```
[root@master ~]# tar -zxvf automake-1.11.tar.gz
[root@master ~]# cd automake-1.11
[root@master automake-1.11]# ./configure
[root@master automake-1.11]#make
[root@master automake-1.11]#make install
```

再次执行：

```
[root@master ClusterLabs-resource-agents-b735277]# ./autogen.sh
...
configure.ac:82: installing './compile'
configure.ac:33: installing './config.guess'
configure.ac:33: installing './config.sub'
configure.ac:63: installing './install-sh'
configure.ac:63: installing './missing'
heartbeat/Makefile.am: installing './depcomp'
autoreconf: Leaving directory '.'
```

```
Now run ./configure and make
[root@master ClusterLabs-resource-agents-b735277]# ./configure

…

resource-agents configuration:
  Version  = UNKNOWN
  Build Version= e2619435b0758c707a70b9cfb6cea836f6b091b8
  Features =

  Prefix   = /usr
  Executables  = /usr/sbin
  Man pages= /usr/share/man
  Libraries= /usr/lib
  Header files = /usr/include
  Arch-independent files   = /usr/share
  Documentation= /usr/share/doc/resource-agents
  State information= /usr/var
  System configuration = /usr/etc
  RA state files   = /usr/var/run/resource-agents
  AIS Plugins  =
CFLAGS=-g-O2-ggdb3-fgnu89-inline-fstack-protector-all-Wall-Wbad-function
-cast-Wcast-qual-Wcast-align-Wdeclaration-after-statement-Wendif-labels-Wflo
at-equal-Wformat=2-Wformat-security-Wformat-nonliteral-Winline-Wmissing-prot
otypes-Wmissing-declarations-Wmissing-format-attribute-Wnested-externs-Wno-l
ong-long-Wno-strict-aliasing-Wpointer-arith-Wstrict-prototypes-Wwrite-string
s-ansi-D GNU SOURCE-DANSI ONLY-Werror
  Libraries=  -L/lib -lglib-2.0
  Stack Libraries  =
[root@master ClusterLabs-resource-agents-b735277]#make
[root@master ClusterLabs-resource-agents-b735277]#make install
```

至此，安装完毕。

9.2 配置 Heartbeat

相关目录/etc/ha.d/：

```
[root@master ha.d]# tree /etc/ha.d/
/etc/ha.d/
|-- README.config
|-- harc
|-- rc.d
|   |-- ask_resources
```

```
|   |-- hb_takeover
|   |-- ip-request
|   |-- ip-request-resp
|   '-- status
'-- resource.d
|-- AudibleAlarm
|-- Delay
|-- Filesystem
|-- ICP
|-- IPaddr
|-- IPaddr2
|-- IPsrcaddr
|-- IPv6addr
|-- LVM
|-- LinuxSCSI
|-- MailTo
|-- OCF
|-- Raid1
|-- SendArp
|-- ServeRAID
|-- WAS
|-- WinPopup
|-- Xinetd
|-- apache
|-- db2
|-- hto-mapfuncs
|-- ids
'-- portblock

2 directories, 30 files
```

我们来看一下 README.config 文件：

```
[root@master ha.d]# more README.config
You need three configuration files to make heartbeat happy,
and they all go in this directory.

They are:
ha.cf   Main configuration file
haresources Resource configuration file
authkeysAuthentication information

These first two may be readable by everyone, but the authkeys file
must not be.
```

```
The good news is that sample versions of these files may be found in
the documentation directory (providing you installed the documentation).

If you installed heartbeat using rpm packages then
this command will show you where they are on your system:
rpm -q heartbeat -d

If you installed heartbeat using Debian packages then
the documentation should be located in /usr/share/doc/heartbeat
```

根据这个文件的意思，我们需要使用三个文件，即 ha.cf、haresources 和 authkeys，另外还有个条件，对于 ha.cf、haresources 文件的权限需要设置为任何人都可读，而 authkeys 则只能是属主可读。

可以通过找到源安装包中包括的示例配置文件来修改它们以便达到具体的使用：

```
[root@master doc]# pwd
/root/heartbeat/Heartbeat-3-0-7e3a82377fa8/doc
[root@master doc]# cp ha.cf  haresources  authkeys  /etc/ha.d/
[root@master doc]# chmod 0600 /etc/ha.d/authkeys
```

9.2.1 ha.cf 文件

该配置文件中有很多选项，下面我们先来了解一下这些选项。

1. 单位设定

关于时间值

在配置文件中，涉及时间的选项，默认时间单位为秒（second），例如 10 就是 10 秒的意思，也可以使用毫秒（millisecond）作为单位，例如 1500ms 就是 1.5 秒。

关于布尔值

任何下面设置的值（注意不区分大小写），它们都表示为真：

```
true, on, yes, y, 1
```

任何下面设置的值（注意不区分大小写），它们都表示为假：

```
false, off, no, n, 0
```

2. 选项

配置 ha.cf 提供的选项如下。

选项名称：debugfile

功能：该选项用于设置调试日志。

在日志记录设置方面要注意以下事项：如果 debugfile、logfile 和 logfacility 都没有定义，那么日志记录就相当于设置为“use_logd yes”，否则它们将各自生效。如果想要阻止记录日志到 syslog，那么 logfacility 必须设置为“none”。

例如：debugfile /var/log/ha-debug

选项名称：logfile

默认值：logfile/var/log/ha-log

功能：该选项用于设置 Heartbeat 日志文件在系统中的位置。

选项名称：logfacility

默认值：logfacility local0

功能：该选项用于设置 syslog（）/logger 设备。

选项名称：keepalive

默认值：2

功能：指定心跳间隔时间。

选项名称：deadtime

默认值：30

功能：指定备用机器在该选项指定的时间内如果没有收到主节点的心跳信息，那么就宣布接管主服务器的资源。不要将该值设置得太低。

选项名称：warntime

默认值：10

功能：该选项用于设置警告时间。

选项名称：initdead

默认值：120

功能：在某些机器或者操作系统上，网络启动需要一定的时间，因此在重新启动系统后要等到网络正常工作后。该选项用于设置网络能正常开始工作的时间，其值应该是 deadtime 选项的两倍。换句话说，就是在网络重启多少秒后才开始计算心跳，这种做法的意义是为了防止在网络还没有初始化完毕之后就已经到了心跳死亡的时间，这样容易造成错误认定死亡的结果。

选项名称：udpport

默认值：694

功能：该选项用于设置使用 bcast/ucast 通信时所使用的端口。

选项名称：baud

默认值：19200

功能：该选项用于设置串行口的波特率。

选项名称：serial

功能：该选项用于设置串行端口的名称。

例如：

```
serial /dev/ttyS0  # Linux
serial /dev/cuaa0  # FreeBSD
serial /dev/cuad0  # FreeBSD 6.x
serial /dev/cua/a  # Solaris
```

选项名称：bcast

功能：该选项用于设置 heartbeat 在哪个网卡上监听心跳的广播信息。

例如：

```
bcast  eth0# Linux
bcast  eth1 eth2  # Linux
bcast  le0 # Solaris
bcast  le1 le2 # Solaris
```

选项名称：mcast

默认值：mcast eth0 225.0.0.1 694 1 0

语法格式：mcast [dev] [mcast group] [port] [ttl] [loop]

- [dev]：设置接收、发送心跳的设备；
- [mcast group]：设置加入的多播组（是一个 D 类的组播地址 224.0.0.0 ～ 239.255.255.255）；
- [port]：设置用于发送、接收数据包的 UDP 端口；
- [ttl]：设置 TTL 值；
- [loop]：是否设置为设备。

功能：该选项用于设置 Heartbeat 多播。

选项名称：ucast

语法格式：ucast [dev] [peer-ip-addr]

- [dev]：设置用于接收、发送数据包的设备；
- [peer-ip-addr]：设置对方机器用于发送数据包的 IP 地址。

功能：该选项用于设置 UDP 单播方式。

例如：

```
ucast eth0 192.168.1.2
```

选项名称：auto_failback

默认值：on

功能：该选项用于设置当主节点恢复正常后是否自动切换回服务（即主节点仍然是主节点，备份节点仍然是备份节点）。

该选项有以下三种取值。

- on：自动切换回服务；
- off：不自动切换；
- legacy：在系统中所有的节点还都不支持 auto_failback 选项时，使用该值则会自动切换回服务。

值 on 和 off 用于向下兼容旧的选项 nice_failback on 的设置。

选项名称：stonith

语法格式：stonith <stonith_type> <configfile>

功能：该选项用于对 stonith 设备设置，就是说如果在集群中有 stonith 设备，那么可以通过

该设备对其进行设置。

例如：

```
stonith baytech /etc/ha.d/conf/stonith.baytech
```

选项名称：stonith_host

语法格式：stonith_host <hostfrom> <stonith_type> <params...>

功能：可以通过该选项来设置多个 stonith 设备。

例如：

```
stonith_host * baytech 10.0.0.3 mylogin mysecretpassword
stonith_host ken3  rps10 /dev/ttyS1 kathy 0
stonith_host kathy rps10 /dev/ttyS1 ken3 0
```

选项名称：watchdog

默认值：/dev/watchdog

功能：该选项的功能在于通过 heartbeat 来监视系统。

选项名称：node

语法格式：nodenodename ... （这里的节点名必须使用 uname -n 取得）

功能：该选项用于告诉在集群中的机器。

例如：

```
node   ken3
node   Kathy
```

以下选项则很少用到。

选项名称：ping

功能：该选项会 ping 一个设备的 IP 地址，它将设备看做集群的伪设备。该选项会和 ipfail 一同使用。注意，千万不要使用集群中的任何一个节点。

例如：

```
ping 10.10.10.254
```

选项名称：ping_group

功能：设置 ping 组，例如将 10.10.10.254 和 10.10.10.253 作为一个集群中的伪设备，称之为 group1，如果这两者中的任何一个被启用，那么 group1 将会被启用。同样它和 ipfail 一同工作。

例如：

```
ping_group group1 10.10.10.254 10.10.10.253
```

选项名称：hbaping

功能：该选项用于光纤通道适配器 HBA 的 ping 指令，将 fc-card-name（光纤通道卡名字）看做是集群中的一个伪设备。可以从 http://hbaapi.sourceforge.net 获取 HBAAPI。

例如：

```
hbaping qlogic-qla2200-0
```

选项名称：respawn

功能：该选项用于设置被 respawn 监控的命令，该进程会随着 heartbeat 一同启动或者是停止，除非它们通过 rc=100 退出，才会重新启动。

例如：

```
respawn userid /path/name/to/run
respawn hacluster /usr/lib/heartbeat/ipfail
```

选项名称：apiauth

功能：该选项用于设置客户端 API 访问控制。默认为禁止访问。

例如：

```
apiauth client-name gid=gidlist uid=uidlist
apiauth ipfail gid=haclient uid=hacluster
```

选项名称：hopfudge

功能：该选项用于设置集群中的总共跳数，该参数用于环型网络的设置。

选项名称：deadping

默认值： 30

功能：该选项用于设置 ping 节点的死亡时间。

选项名称：hbgenmethod

功能：该选项用于设置 heartbeat 产生数字的创建方法。通常数字被存储在磁盘上并且会根据需要再进行增加。

例如：

```
hbgenmethod time
```

选项名称：realtime

默认值：on

功能：启用或者禁用实时执行（高优先级）。

选项名称：debug

默认值：0

功能：该选项用于设置调试级别。

选项名称：apiauth

功能：该选项用于设定 API 认证，用于替代以前的基于文件系统管道权限认证方法。可以指定一个 uid 或/和 gid 列表，如果指定了这两者，那么符合 uid 列表或者符合 gid 列表中的进程都将通过验证。

组名称“default”有特殊的含义，如果指定了它，那么它将用于验证无组的客户端和任何没有另外指定的客户组。

下面是例外情况，“default”不能用于以下情况，括号中指出实际的认证指令：

```
ipfail (uid=HA_CCMUSER)
ccm    (uid=HA_CCMUSER)
ping   (gid=HA_APIGROUP)
cl_status  (gid=HA_APIGROUP)
```

例如：

```
apiauth ipfail uid=hacluster
apiauth ccm uid=hacluster
apiauth cms uid=hacluster
apiauth ping gid=haclient uid=alanr,root
apiauth default gid=haclient
```

选项名称：msgfmt

默认值：classic

功能：该选项用于设置消息在介质中传输时的格式，可选择的值有 classic 和 netstring。

选项名称：use_logd

功能：该选项用设置是否使用日志守护进程。如果使用了日志守护进程，那么该文件中 logfile、debugfile 和 logfacility 选项的设置将不再有意义。如果使用这种方式，那么要检查它的配置文件（默认位置为/etc/logd.cf）。推荐将该选项设置为“yes”。

配置文件 logd.cf 在 glue 的安装包（logd/logd.cf）中，通过复制并进行适当修改就可以了。

选项名称：conn_logd_time

默认值：60

功能：该选项用于先前连接日志守护进程失败后，设置重新连接日志守护进程的间隔。

选项名称：compression

功能：该选项用于配置压缩模块，可选择的值有 zlib 或者 bz2，这依赖于系统中具体的库。

例如：

```
compressionbz2
```

选项名称：compression_threshold

默认值：2

功能：该选项用于配置压缩门限值。例如，设置为 1，表示压缩门限值为 1KB，任何大于 1KB 的信息都将会被压缩后传输。

例如：

```
compression_threshold 1
```

3. 配置内容

下面是 ha.cf 文件的配置内容，主、备节点都一样：

```
[root@master ~]# more /usr/etc/ha.d/ha.cf
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility local0
keepalive 2
deadtime 30
warntime 10
initdead 30
udpport 694
bcast eth1 eth0
```

```
auto_failback on
nodemaster
nodeslave
```

9.2.2 haresources 文件

haresources 文件由 heartbeat 在启动时调用，它比较简单，就是对资源的管理，在我们的使用中仅仅是让它提供一个“浮动”的 IP 地址。

特别注意：文件 haresources 在集群中所有节点上都必须相同。

在该文件中添加以下条目（主、备节点都一样）：

```
master  IPaddr::192.168.3.2/24/eth0:1/
```

9.2.3 authkeys 文件

authkeys 文件是一个认证文件（主、备节点都一样）：

```
[root@master ~]# more /usr/etc/ha.d/authkeys
auth 1
#1 crc
1 sha1 HI!
#3 md5 Hello!
```

要求该文件的权限是 600。

9.3 启动 Heartbeat

在安装与配置完成 Heartbeat 之后，包括主节点和备用节点的安装和配置，启动 Heartbeat 服务。正常情况下按照先启动主服务器，然后再启动备用服务器。

9.3.1 环境部署

下面的部署为 Linux 系统环境的部署，即 IP 地址的分配与设定。

1. 系统环境部署

环境部署中主要有以下三个需要设置的地方。

第一个需要设置的地方是服务器的 IP 地址

在 Linux 服务器上 IP 地址的设置如下。

Master 服务器:

```
eth0：192.168.3.198
eth1：192.168.9.7
```

Slave 服务器:

```
eth0：192.168.3.198
eth1：192.168.9.8
```

VIP（Nginx 提供服务器的 IP）：

```
eth0:1：192.168.3.2
```

> 需要注意的一点是，这个浮动的 VIP 是不能设置为静态 IP 地址的，它是由 Heartbeat 控制的一个资源生成。

第二个要设置的地方是 hosts 文件

由于 Heartbeat 在设置时使用了名字解析，因此要在 hosts 文件中添加相应的节点名字。节点的名字要使用“uname -n”来获取，例如：

```
[root@master ~]# uname -n
master
```

在主、备节点的 hosts 文件中添加以下内容：

```
[root@master ~]# cat /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1   master localhost.localdomain localhost
::1 localhost6.localdomain6 localhost6
192.168.3.198   master
192.168.9.199   slave
```

第三个需要设置的地方

我们在安装 Heartbeat 时没有使用--prefix 选项锁定目录，当使用/etc/init.d/heartbeat 脚本启动 Heartbeat 服务器时会出现问题，因此需要执行以下操作（如果你定制了安装目录，那么按照你自己的定制来操作）：

```
[root@master ~]# cd /etc/ha.d/
[root@master ~]# cp -R ./* /usr/etc/ha.d/
```

在主、备服务器上都要执行复制操作。

2. Nginx 访问文件部署

在设定 Nginx 提供的访问文件中，分为生产环境和实验环境两种情况。在实验环境调好之后即可切换为生产环境，其实就是更高 Nginx 的配置文件。

生产环境

对于实际应用中的服务器部署，必须在主节点和备用节点上都安装 Nginx 服务，并且将其启动，不依赖于 Heartbeat 启动 Nginx 服务。另外，对于主节点和备用节点上的 Nginx 配置也必须完全一样。

实验环境

为了查看实际的效果，我们采用了这个实验环境。在这里我们将主节点和备用节点的主页设置为不同的内容，这样就能明确地看出它们之间的切换情况。

主节点的主页内容：

```
[root@master ~]# cat /usr/local/nginx-1.0.2/html/index.html
```

```
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Master!</h1></center>
</body>
</html>
```

备用节点的主页内容：

```
[root@slave ~]# cat /usr/local/nginx-1.0.2/html/index.html
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Slave!</h1></center>
</body>
</html>
```

9.3.2 启动主 Heartbeat

在设置 Heartbeat 在 Linux 服务器启动时的启动级别时，可以根据实际情况来设定启动级别。在较高质量的电源环境中，我们使用手动启动 Heartbeat 服务更好。

1. 必要设置

在安装 Heartbeat 服务时会在/etc/init.d/目录下生成一个 Heartbeat 的启动脚本。以下是它的执行情况：

```
[root@master ~]# ll /etc/init.d/heartbeat
-rwxr-xr-x 1 root root 10819 Nov 15 19:25 /etc/init.d/heartbeat
[root@master ~]# chkconfig --list heartbeat
service heartbeat supports chkconfig, but is not referenced in any runlevel
(run 'chkconfig --add heartbeat')
[root@master ~]# chkconfig --list|grep  heartbeat
[root@master ~]#
```

这个现象告诉我们，虽然 Heartbeat 脚本在/etc/init.d/目录下，但却没有设定运行级别，因此我们可以通过以下命令来完成：

```
[root@master ~]# chkconfig --add heartbeat
[root@master ~]# chkconfig --list|grep  heartbeat
heartbeat    0:off   1:off   2:on3:on4:on5:on6:off
```

使用--add 选项的默认添加级别是 2345，如果你有特殊需求的话，可以通过--levels 选项添加。

在备份服务器上进行同样的设定操作。

2. 启动 Heartbeat

可以通过“service heartbeat start”命令或“/etc/init.d/heartbeat start”来启动 Heartbeat 服务：

```
[root@master ~]# /etc/init.d/heartbeat start
Starting High-Availability services: IPaddr[11984]: INFO:  Resource is stopped
   [  OK  ]
```

查看进程：

```
[root@master ~]# ps -ef|grep heartbeat
root 13515 1  0 09:23 ?  00:00:00  heartbeat: master control process
root 13518 13515  0 09:23 ?   00:00:00 heartbeat: FIFO reader
root 13519 13515  0 09:23 ?   00:00:00 heartbeat: write: bcast eth1
root 13520 13515  0 09:23 ?   00:00:00 heartbeat: read: bcast eth1
root 13521 13515  0 09:23 ?   00:00:00 heartbeat: write: bcast eth0
root 13522 13515  0 09:23 ?   00:00:00 heartbeat: read: bcast eth0
```

同时监控日志：

```
Nov 18 08:21:53 master heartbeat: [12043]: info: Pacemaker support: false
Nov 18 08:21:53 master heartbeat: [12043]: WARN: Logging daemon is disabled --enabling logging daemon is recommended
Nov 18 08:21:53 master heartbeat: [12043]: info: **************************
Nov 18 08:21:53 master heartbeat: [12043]: info: Configuration validated. Starting heartbeat 3.0.5
Nov 18 08:21:53 master heartbeat: [12044]: info: heartbeat: version 3.0.5
Nov 18 08:21:53 master heartbeat: [12044]: info: Heartbeat generation: 1321397217
Nov 18 08:21:53 master heartbeat: [12044]: info: glib: UDP Broadcast heartbeat started on port 694 (694) interface eth1
Nov 18 08:21:53 master heartbeat: [12044]: info: glib: UDP Broadcast heartbeat closed on port 694 interface eth1 - Status: 1
Nov 18 08:21:53 master heartbeat: [12044]: info: glib: UDP Broadcast heartbeat started on port 694 (694) interface eth0
Nov 18 08:21:53 master heartbeat: [12044]: info: glib: UDP Broadcast heartbeat closed on port 694 interface eth0 - Status: 1
Nov 18 08:21:53 master heartbeat: [12044]: info: Local status now set to: 'up'
Nov 18 08:21:54 master heartbeat: [12044]: info: Link master:eth1 up.
Nov 18 08:21:54 master heartbeat: [12044]: info: Link master:eth0 up.
Nov 18 08:22:23 master heartbeat: [12044]: WARN: node slave: is dead
Nov 18 08:22:23 master heartbeat: [12044]: info: Comm_now_up(): updating status to active
Nov 18 08:22:23 master heartbeat: [12044]: info: Local status now set to: 'active'
```

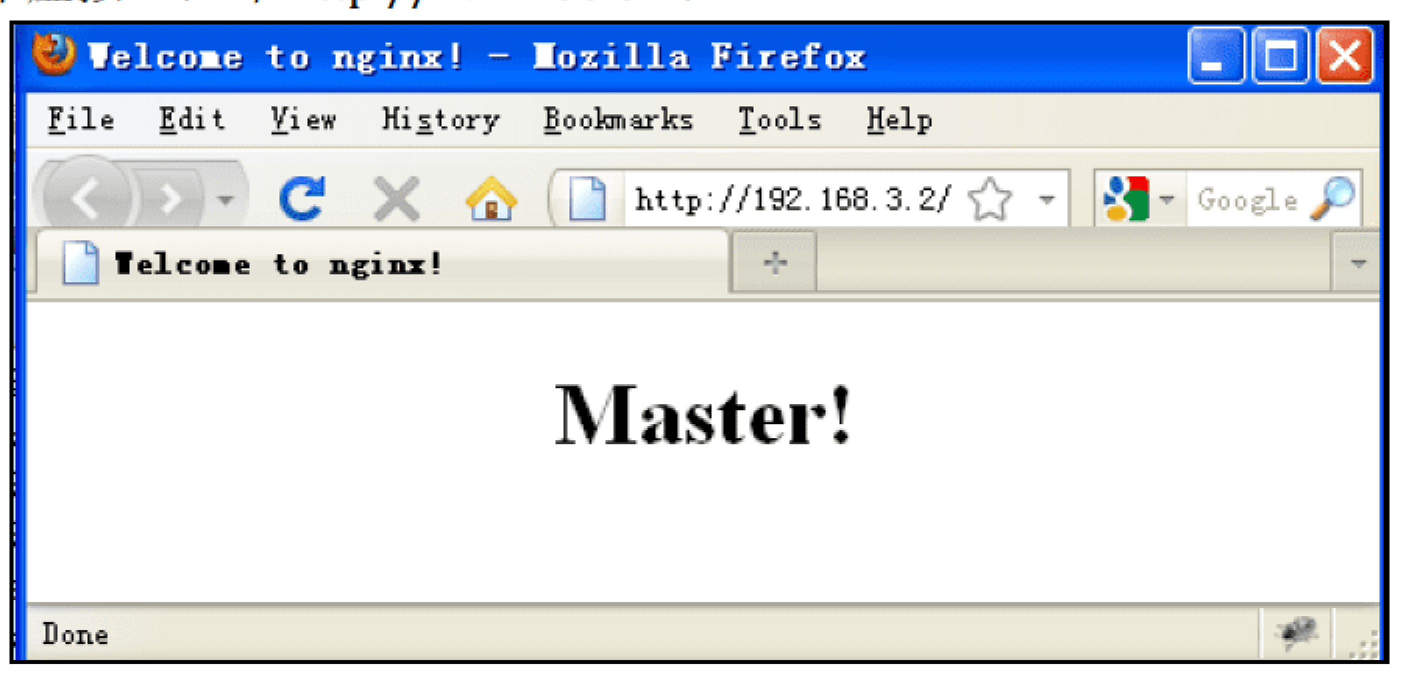
```
Nov 18 08:22:23 master heartbeat: [12044]: WARN: No STONITH device configured.
Nov 18 08:22:23 master heartbeat: [12044]: WARN: Shared disks are not protected.
Nov 18 08:22:23 master heartbeat: [12044]: info: Resources being acquired from slave.
Nov18 08:22:23 master harc[12055]: info: Running /usr/etc/ha.d//rc.d/ status status
Nov18 08:22:23mastermach_down[12108]:info:/usr/share/heartbeat/mach_down: nice_failback: foreign resources acquired
Nov 18 08:22:23 master mach_down[12108]: info: mach_down takeover complete for node slave.
Nov 18 08:22:24 master IPaddr[12123]: INFO:  Resource is stopped
Nov 18 08:22:24 master heartbeat: [12056]: info: Local Resource acquisition completed.
Nov 18 08:22:24 master heartbeat: [12044]: info: mach_down takeover complete.
Nov 18 08:22:24 master heartbeat: [12044]: info: Initial resource acquisition complete (mach_down)
Nov18 08:22:24masterharc[12187]:info:Running/usr/etc/ha.d//rc.d/ip-request-resp ip-request-resp
Nov 18 08:22:24 master ip-request-resp[12187]: received ip-request-resp IPaddr::192.168.3.2/24/eth0:1 OK yes
Nov 18 08:22:24 master ResourceManager[12210]: info: Acquiring resource group: master IPaddr::192.168.3.2/24/eth0:1
Nov 18 08:22:24 master IPaddr[12238]: INFO:  Resource is stopped
Nov18 08:22:24masterResourceManager[12210]:info:Running/etc/ha.d/resource.d/IPaddr 192.168.3.2/24/eth0:1 start
Nov 18 08:22:24 master IPaddr[12323]: INFO: Using calculated netmask for 192.168.3.2: 255.255.255.0
Nov 18 08:22:24 master IPaddr[12323]: INFO: eval ifconfig eth0:3 192.168.3.2 netmask 255.255.255.0 broadcast 192.168.3.255
Nov 18 08:22:24 master avahi-daemon[2775]: Registering new address record for 192.168.3.2 on eth0.
Nov 18 08:22:24 master IPaddr[12297]: INFO:  Success
Nov 18 08:22:34 master heartbeat: [12044]: info: Local Resource acquisition completed. (none)
Nov 18 08:22:34 master heartbeat: [12044]: info: local resource transition completed.
```

正如我们看到的，日志分为以下两部分：

第一部分是主 Heartbeat 启动的日志，第二部分则是检测备份节点并载入资源的日志。在这里需要明确的一点是备份节点 dead，即备份节点并没有启动。

3. 访问测试

我们访问这个虚拟 IP，即 http://192.168.3.2：



没错，访问的是主服务器的页面。

9.3.3 启动备用 Heartbeat

1. 必要设置

同 9.3.2 小节的“启动主 Heartbeat”。

2. 启动 Heartbeat

同样可以通过“service heartbeat start”命令或“/etc/init.d/heartbeat start”来启动 Heartbeat 服务：

```
[root@master ~]# /etc/init.d/heartbeat start
Starting High-Availability services: IPaddr[11984]: INFO:  Resource is
stopped
   [  OK  ]
[root@slave ~]# /etc/init.d/heartbeat start
Starting High-Availability services: IPaddr[16753]: INFO:  Resource is
stopped
   [  OK  ]
```

同时监控日志：

```
Nov 18 08:46:07 slave heartbeat: [16812]: info: Pacemaker support: false
Nov 18 08:46:07 slave heartbeat: [16812]: WARN: Logging daemon is disabled
--enabling logging daemon is recommended
Nov 18 08:46:07 slave heartbeat: [16812]: info: **************************
Nov 18 08:46:07 slave heartbeat: [16812]: info: Configuration validated.
Starting heartbeat 3.0.5
Nov 18 08:46:07 slave heartbeat: [16813]: info: heartbeat: version 3.0.5
Nov 18 08:46:07 slave heartbeat: [16813]: info: Heartbeat generation:
```

```
1321516460
    Nov 18 08:46:07 slave heartbeat: [16813]: info: glib: UDP Broadcast heartbeat started on port 694 (694) interface eth1
    Nov 18 08:46:07 slave heartbeat: [16813]: info: glib: UDP Broadcast heartbeat closed on port 694 interface eth1 - Status: 1
    Nov 18 08:46:07 slave heartbeat: [16813]: info: Local status now set to: 'up'
    Nov 18 08:46:08 slave heartbeat: [16813]: info: Link slave:eth1 up.
    Nov 18 08:46:08 slave heartbeat: [16813]: info: Link master:eth1 up.
    Nov 18 08:46:08 slave heartbeat: [16813]: info: Status update for node master: status active
    Nov 18 08:46:08 slave harc[16820]: info: Running /usr/etc/ha.d//rc.d/status status
    Nov 18 08:46:09 slave heartbeat: [16813]: info: Comm_now_up() : updating status to active
    Nov 18 08:46:09 slave heartbeat: [16813]: info: Local status now set to: 'active'
    Nov 18 08:46:09 slave heartbeat: [16813]: info: remote resource transition completed.
    Nov 18 08:46:09 slave heartbeat: [16813]: info: remote resource transition completed.
    Nov 18 08:46:09 slave heartbeat: [16813]: info: Local Resource acquisition completed. (none)
    Nov 18 08:46:10 slave heartbeat: [16813]: info: master wants to go standby [foreign]
    Nov 18 08:46:10 slave heartbeat: [16813]: info: standby: acquire [foreign] resources from master
    Nov 18 08:46:10 slave heartbeat: [16840]: info: acquire local HA resources (standby).
    Nov 18 08:46:10 slave heartbeat: [16840]: info: local HA resource acquisition completed (standby).
    Nov 18 08:46:10 slave heartbeat: [16813]: info: Standby resource acquisition done [foreign].
    Nov 18 08:46:10 slave heartbeat: [16813]: info: Initial resource acquisition complete (auto_failback)
    Nov 18 08:46:11 slave heartbeat: [16813]: info: remote resource transition completed.
```

同时监控主服务器。在主服务器上报告以下内容：

```
    Nov 18 08:46:11 master heartbeat: [12044]: info: Link slave:eth1 up.
    Nov 18 08:46:11 master heartbeat: [12044]: info: Status update for node slave: status init
    Nov 18 08:46:11 master heartbeat: [12044]: info: Status update for node slave: status up
```

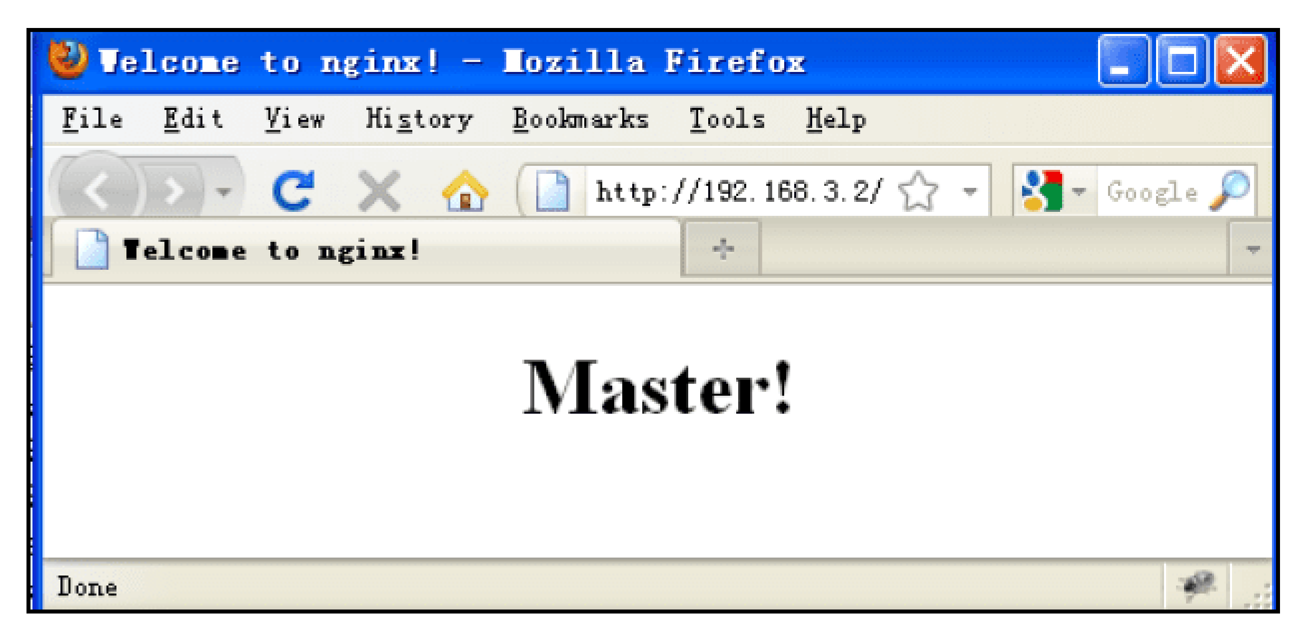
```
    Nov18 08:46:11 master harc[12455]: info: Running /usr/etc/ha.d//rc.d/
status status
    Nov18 08:46:11 master harc[12473]: info: Running /usr/etc/ha.d//rc.d/
status status
    Nov 18 08:46:11 master heartbeat: [12044]: info: all clients are now paused
    Nov 18 08:46:12 master heartbeat: [12044]: info: Status update for node
slave: status active
    Nov18 08:46:12 master harc[12490]: info: Running /usr/etc/ha.d//rc.d/
status status
    Nov 18 08:46:12 master heartbeat: [12044]: info: remote resource transition
completed.
    Nov 18 08:46:12 master heartbeat: [12044]: info: master wants to go standby
[foreign]
    Nov 18 08:46:13 master heartbeat: [12044]: info: standby: slave can take
our foreign resources
    Nov 18 08:46:13 master heartbeat: [12507]: info: give up foreign HA resources
（standby）.
    Nov 18 08:46:13 master heartbeat: [12507]: info: foreign HA resource release
completed （standby）.
    Nov 18 08:46:13 master heartbeat: [12044]: info: Local standby process
completed [foreign].
    Nov 18 08:46:13 master heartbeat: [12044]: info: all clients are now resumed
    Nov 18 08:46:13 master heartbeat: [12044]: WARN: 1 lost packet(s） for [slave]
[10:12]
    Nov 18 08:46:13 master heartbeat: [12044]: info: remote resource transition
completed.
    Nov 18 08:46:13 master heartbeat: [12044]: info: No pkts missing from slave!
    Nov 18 08:46:13 master heartbeat: [12044]: info: Other node completed standby
takeover of foreign resources.
```

正如日志中报告的，这两个日志合起来的意思如下。

备用服务器在启动后去检查主服务器，发现主服务器是 active，即活跃的，更通俗地说是活的。这时主服务器也坚持到备用服务器“活”过来，想要和它争夺服务“权”。这时主服务器就会根据配置文件中的协定，要求备用服务器保持备用，不要“夺权”，于是备用服务器就根据配置文件（或者叫“协定”）的规定乖乖地保持备用状态。什么时候备用服务器才能有“出头”之日，不要着急，继续往后看。

3. 访问测试

现在我们再次访问这个虚拟 IP，即 http://192.168.3.2：



没错，访问的还是主服务器的页面。

9.4 测试 Heartbeat

当正常启动了 Heartbeat 之后，我们来进行以下测试，以便查看它的健壮性。

9.4.1 宕掉主节点

我们将主节点宕掉：

```
[root@master ~]# pkill heartbeat
```

1. 监控日志

我们同时监控主节点和备用节点的日志。

监控主服务器日志：

```
Nov 18 09:02:55 master heartbeat: [12044]: ERROR: Cannot write to media pipe
0: Resource temporarily unavailable
Nov 18 09:02:55 master heartbeat: [12044]: ERROR: Killing and restarting
communications processes.: Resource temporarily unavailable
Nov 18 09:02:55 master heartbeat: [12044]: info: glib: UDP Broadcast
heartbeat closed on port 694 interface eth1 - Status: 1
Nov 18 09:02:55 master heartbeat: [12044]: ERROR: Cannot write to media pipe
1: Invalid argument
Nov 18 09:02:55 master heartbeat: [12044]: ERROR: Killing and restarting
communications processes.: Invalid argument
Nov 18 09:02:55 master heartbeat: [12044]: info: glib: UDP Broadcast
heartbeat closed on port 694 interface eth0 - Status: 1
Nov 18 09:02:55 master heartbeat: [12044]: CRIT: send_to_all_media: No
working comm channels to write to.
Nov 18 09:02:55 master heartbeat: [12044]: info: Heartbeat shutdown in
progress. (12044)
Nov 18 09:02:55 master heartbeat: [12557]: info: Giving up all HA resources.
Nov 18 09:02:55 master heartbeat: [12044]: info: Core process 12047 exited.
5 remaining
Nov 18 09:02:55 master heartbeat: [12044]: info: Core process 12048 exited.
```

```
4 remaining
    Nov 18 09:02:55 master heartbeat: [12044]: info: Core process 12049 exited.
3 remaining
    Nov 18 09:02:55 master heartbeat: [12044]: info: Core process 12050 exited.
2 remaining
    Nov 18 09:02:55 master heartbeat: [12044]: info: Core process 12051 exited.
1 remaining
    Nov 18 09:02:55 master heartbeat: [12044]: info: master Heartbeat shutdown
complete.
    Nov 18 09:02:55 master ResourceManager[12570]: info: Releasing resource
group: master IPaddr::192.168.3.2/24/eth0:1
    Nov18 09:02:55masterResourceManager[12570]:info:Running /etc/ha.d/
resource.d/IPaddr 192.168.3.2/24/eth0:1 stop
    Nov 18 09:02:55 master IPaddr[12633]: INFO: ifconfig eth0:3 down
    Nov 18 09:02:55 master avahi-daemon[2775]: Withdrawing address record for
192.168.3.2 on eth0.
    Nov 18 09:02:55 master IPaddr[12607]: INFO:  Success
    Nov 18 09:02:55 master heartbeat: [12557]: info: All HA resources
relinquished.
```

监控备用服务器日志：

```
    Nov 18 09:03:17 slave last message repeated 6 times
    Nov 18 09:03:21 slave heartbeat: [16813]: WARN: node master: is dead
    Nov 18 09:03:21 slave heartbeat: [16813]: WARN: No STONITH device configured.
    Nov 18 09:03:21 slave heartbeat: [16813]: WARN: Shared disks are not
protected.
    Nov 18 09:03:21 slave heartbeat: [16813]: info: Resources being acquired
from master.
    Nov 18 09:03:21 slave heartbeat: [16813]: info: Link master:eth1 dead.
    Nov 18 09:03:21 slave harc[16891]: info: Running /usr/etc/ha.d//rc.d/
status status
    Nov 18 09:03:21 slave heartbeat: [16892]: info: No local resources
[/usr/share/heartbeat/ResourceManager listkeys slave] to acquire.
    Nov 18 09:03:21 slave mach down[16921]: info: Taking over resource group
IPaddr::192.168.3.2/24/eth0:1
    Nov 18 09:03:21 slave ResourceManager[16948]: info: Acquiring resource
group: master IPaddr::192.168.3.2/24/eth0:1
    Nov 18 09:03:21 slave IPaddr[16976]: INFO:  Resource is stopped
    Nov 18 09:03:21slaveResourceManager[16948]:info:Running /etc/ha.d/
 resource.d/IPaddr 192.168.3.2/24/eth0:1 start
    Nov 18 09:03:21 slave IPaddr[17061]: INFO: Using calculated netmask for
192.168.3.2: 255.255.255.0
    Nov 18 09:03:21 slave IPaddr[17061]: INFO: eval ifconfig eth0:1 192.168.3.2
netmask 255.255.255.0 broadcast 192.168.3.255
```

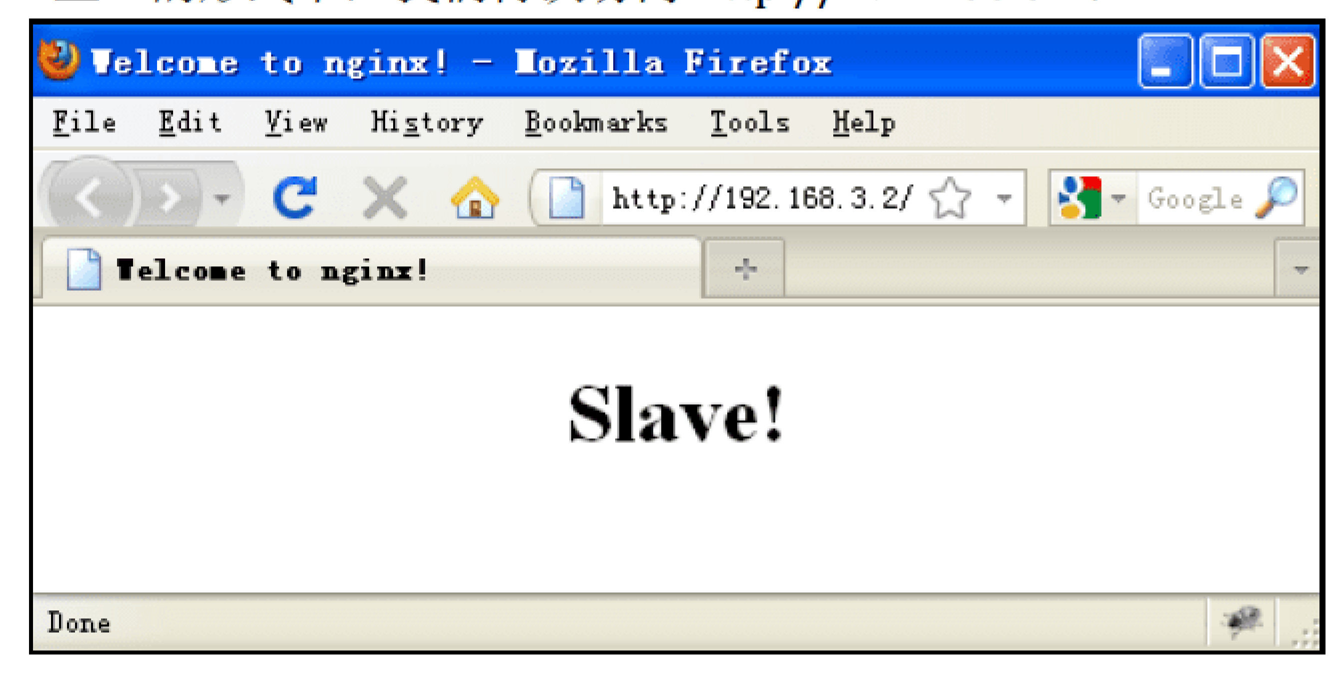
```
Nov 18 09:03:21 slave avahi-daemon[2840]: Registering new address record
for 192.168.3.2 on eth0.
Nov 18 09:03:21 slave IPaddr[17035]: INFO:  Success
Nov 18 09:03:21slavemach down[16921]:info: usr/share/heartbeat/mach down:
nice failback: foreign resources acquired
Nov 18 09:03:21 slave mach down[16921]: info: mach down takeover complete
for node master.
Nov 18 09:03:21 slave heartbeat: [16813]: info: mach_down takeover complete.
Nov 18 09:03:26 slave avahi-daemon[2840]: Invalid query packet.
```

主服务器的日志和备用服务器的日志大致的意思如下。

在主服务器"临终"之前，释放了所有的权力（就是我们所说的资源），但它并没有说明将资源给谁。由于我们采用的是主-备方式，因此作为备用的 Slave 服务器时刻监视着主服务器的一举一动，一旦主服务器宕掉，它马上接替主节点的全部工作，主服务器宕掉了，机会来了，备用服务器接管了所有资源。

2. 访问测试

在主"宕"备"上"的形式下，我们再次访问 http://192.168.3.2：



没错，"变天"了！

9.4.2 重新启动主节点

下面我们再次启动主节点。

1. 监控日志

在启动主节点时我们同时监控主服务器和备用服务器的日志。

主服务器日志：

```
Nov 18 09:23:09 master heartbeat: [13514]: info: Pacemaker support: false
Nov 18 09:23:09 master heartbeat: [13514]: WARN: Logging daemon is disabled
--enabling logging daemon is recommended
Nov 18 09:23:09 master heartbeat: [13514]: info: **************************
Nov 18 09:23:09 master heartbeat: [13514]: info: Configuration validated.
Starting heartbeat 3.0.5
Nov 18 09:23:09 master heartbeat: [13515]: info: heartbeat: version 3.0.5
```

```
    Nov 18 09:23:09 master heartbeat: [13515]: info: Heartbeat generation:
1321397220
    Nov 18 09:23:09 master heartbeat: [13515]: info: glib: UDP Broadcast
heartbeat started on port 694 (694) interface eth1
    Nov 18 09:23:09 master heartbeat: [13515]: info: glib: UDP Broadcast
heartbeat closed on port 694 interface eth1 - Status: 1
    Nov 18 09:23:09 master heartbeat: [13515]: info: glib: UDP Broadcast
heartbeat started on port 694 (694) interface eth0
    Nov 18 09:23:09 master heartbeat: [13515]: info: glib: UDP Broadcast
heartbeat closed on port 694 interface eth0 - Status: 1
    Nov 18 09:23:09 master heartbeat: [13515]: info: Local status now set to:
'up'
    Nov 18 09:23:11 master heartbeat: [13515]: info: Link slave:eth1 up.
    Nov 18 09:23:11 master heartbeat: [13515]: info: Status update for node
slave: status active
    Nov 18 09:23:11 master heartbeat: [13515]: info: Link master:eth0 up.
    Nov 18 09:23:11 master heartbeat: [13515]: info: Link master:eth1 up.
    Nov 18 09:23:11 master harc[13524]: info: Running /usr/etc/ha.d//rc.d/
 status status
    Nov 18 09:23:11 master heartbeat: [13515]: info: Comm_now_up () : updating
status to active
    Nov 18 09:23:11 master heartbeat: [13515]: info: Local status now set to:
'active'
    Nov 18 09:23:12 master heartbeat: [13515]: info: remote resource transition
completed.
    Nov 18 09:23:12 master heartbeat: [13515]: info: remote resource transition
completed.
    Nov 18 09:23:12 master heartbeat: [13515]: info: Local Resource acquisition
completed. (none)
    Nov 18 09:23:12 master heartbeat: [13515]: info: slave wants to go standby
[foreign]
    Nov 18 09:23:13 master heartbeat: [13515]: info: standby: acquire [foreign]
resources from slave
    Nov 18 09:23:13 master heartbeat: [13544]: info: acquire local HA resources
 (standby) .
    Nov 18 09:23:13 master ResourceManager[13557]: info: Acquiring resource
group: master IPaddr::192.168.3.2/24/eth0:1
    Nov 18 09:23:13 master IPaddr[13585]: INFO:  Resource is stopped
    Nov 18 09:23:13 master ResourceManager[13557]: info: Running
/etc/ha.d/resource.d/IPaddr 192.168.3.2/24/eth0:1 start
    Nov 18 09:23:13 master IPaddr[13670]: INFO: Using calculated netmask for
192.168.3.2: 255.255.255.0
    Nov 18 09:23:13 master IPaddr[13670]: INFO: eval ifconfig eth0:3 192.168.3.2
```

```
netmask 255.255.255.0 broadcast 192.168.3.255
    Nov 18 09:23:13 master avahi-daemon[2775]: Registering new address record
for 192.168.3.2 on eth0.
    Nov 18 09:23:13 master IPaddr[13644]: INFO:  Success
    Nov 18 09:23:13 master heartbeat: [13544]: info: local HA resource
acquisition completed (standby).
    Nov 18 09:23:13 master heartbeat: [13515]: info: Standby resource
acquisition done [foreign].
    Nov 18 09:23:13 master heartbeat: [13515]: info: Initial resource
acquisition complete (auto_failback)
    Nov 18 09:23:13 master heartbeat: [13515]: info: remote resource transition
completed.
```

备用服务器日志：

```
    Nov 18 09:23:07 slave heartbeat: [16813]: info: Heartbeat restart on node
master
    Nov 18 09:23:07 slave heartbeat: [16813]: info: Link master:eth1 up.
    Nov 18 09:23:07 slave heartbeat: [16813]: info: Status update for node master:
status init
    Nov 18 09:23:07 slave heartbeat: [16813]: info: Status update for node master:
status up
    Nov 18 09:23:07 slave harc[18213]: info: Running /usr/etc/ha.d//rc.d/status
status
    Nov 18 09:23:07 slave harc[18230]: info: Running /usr/etc/ha.d//rc.d/status
status
    Nov 18 09:23:08 slave heartbeat: [16813]: info: Status update for node master:
status active
    Nov 18 09:23:08 slave harc[18247]: info: Running /usr/etc/ha.d//rc.d/status
status
    Nov 18 09:23:09 slave heartbeat: [16813]: info: remote resource transition
completed.
    Nov 18 09:23:09 slave heartbeat: [16813]: info: slave wants to go standby
[foreign]
    Nov 18 09:23:09 slave heartbeat: [16813]: info: standby: master can take our
foreign resources
    Nov 18 09:23:09 slave heartbeat: [18264]: info: give up foreign HA resources
(standby).
    Nov 18 09:23:09 slave ResourceManager[18277]: info: Releasing resource group:
master IPaddr::192.168.3.2/24/eth0:1
    Nov 18 09:23:09 slave ResourceManager[18277]: info: Running
/etc/ha.d/resource.d/IPaddr 192.168.3.2/24/eth0:1 stop
    Nov 18 09:23:09 slave IPaddr[18340]: INFO: ifconfig eth0:1 down
    Nov 18 09:23:09 slave avahi-daemon[2840]: Withdrawing address record for
192.168.3.2 on eth0.
```

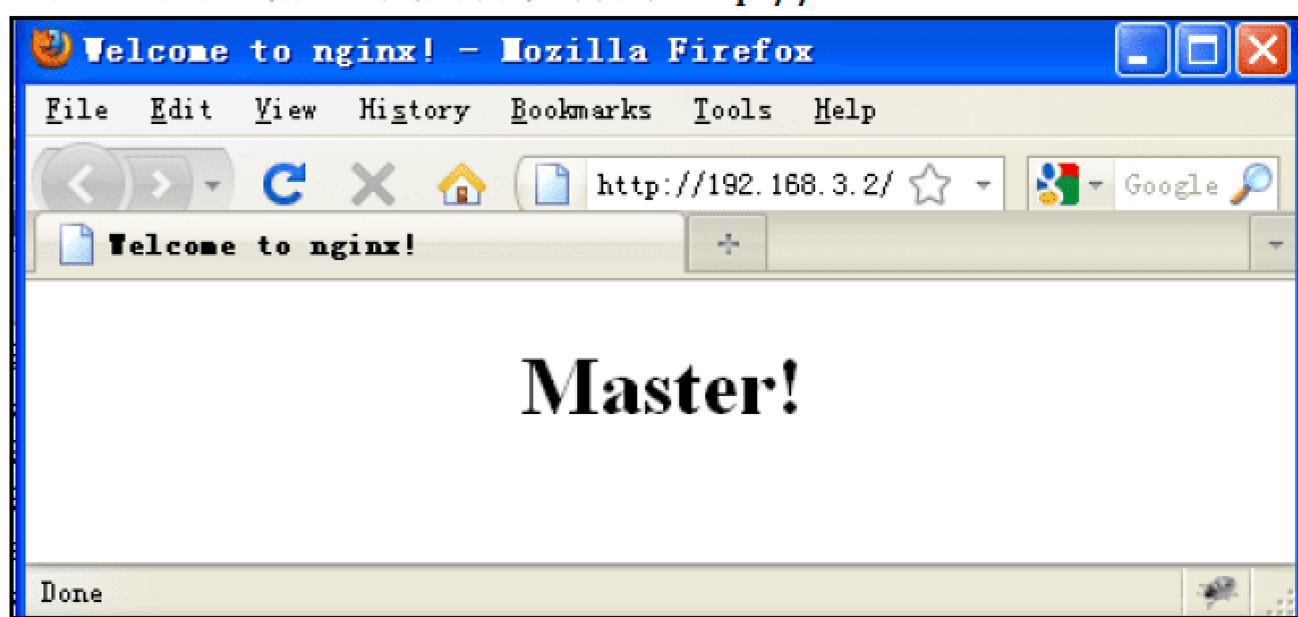
```
    Nov 18 09:23:09 slave IPaddr[18314]: INFO:  Success
    Nov 18 09:23:09 slave heartbeat: [18264]: info: foreign HA resource release
completed (standby).
    Nov 18 09:23:09 slave heartbeat: [16813]: info: Local standby process
completed [foreign].
    Nov 18 09:23:10 slave heartbeat: [16813]: WARN: 1 lost packet(s) for [master]
[10:12]
    Nov 18 09:23:10 slave heartbeat: [16813]: info: remote resource transition
completed.
    Nov 18 09:23:10 slave heartbeat: [16813]: info: No pkts missing from master!
    Nov 18 09:23:10 slave heartbeat: [16813]: info: Other node completed standby
takeover of foreign resources.
```

主服务器的日志和备用服务器的日志大致的意思如下。

正当备用服务器“洋洋得意”的时候，主服务器又奇迹般地“活”过来了，它与行使它权力的备用服务器协商，根据“协议”（就是配置文件）的规定，备用服务器必须交出所有的权力（提供的资源），并且继续保持备用状态。没办法，备用服务器便按照“协议”，保持沉默，“乖乖”地又回到了备用状态。

2. 访问测试

在主“复”备“下”的形式下，我们再次访问 http://192.168.3.2：



没错，又恢复到主服务器的“统治”之下，访问的又是主服务器的页面了。

第 10 章　10 个 QA

10.1 什么是 Nginx

Nginx（发音同 Engine x）是一款轻量级的 Web 服务器 / 反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个 BSD-like 协议下发行。由俄罗斯的程序设计师 Igor Sysoev 开发，最初供俄罗斯大型的入口网站及搜寻引擎 Rambler（俄文：Рамблер）使用。 其特点是：占有内存少、并发能力强。事实上 Nginx 的并发能力在同类型的网页伺服器中表现确实较好。

10.2 Nginx 可以安装在哪些操作系统下

Nginx 在以下的系统上测试并安装过：

- FreeBSD 3 — 8 / i386、FreeBSD 5 — 8 / amd64。
- Linux 2.2 — 2.6 / i386、Linux 2.6 / amd64。
- Solaris 9 / i386、sun4u、Solaris 10 / i386、amd64、sun4v。
- MacOS X / ppc、 i386。
- Windows XP、Windows Server 2003。

10.3 Nginx 在 Windows 下的性能如何

在同等硬件配置下，Nginx 在 Windows 下的性能远低于在 Linux 系统下的性能。

10.4 Nginx 与 Apache 比较有哪些优点

对于这个问题，不能简单地说哪个好，哪个不好，它们各有各的优点。但本人觉得“Apache 重在功能，而 Nginx 重在性能”。简单地阐述一下这个观点：Apache 提供了几百个模块，大家知道，模块就意味着功能，但一个 Apache 服务器最多只有 2000 的并发量；Nginx 提供的模块也就几十个，但它却提供了 20 000 的并发量。Apache 提供的功能多，而 Nginx 提供的功能少，但是有一句话说得很在行：

Apache 就像 Microsoft Word，它有一百万个选项，但你只需要六个。Nginx 只做了这六件事，但是它做的这六件事中有五件事是 Apache 做的 50 倍。

—— Chris Lea，ChrisLea.com

10.5 Nginx 解决了 C10k 问题

为什么 Nginx 能够以出色的性能远远领先于 Apache 的原因，恰好是 Nginx 编写的原因。起初，Igor Sysoev 创建的 Nginx 只是为俄罗斯 Web 站点（www.rambler.ru）提供高流量的访问，每天会收到上亿（hundreds of millions）的请求数量。而 Apache 的设计者早在 20 世纪 90 年代着手设计 Apache 的时候，这很可能不是 Apache 设计师原计划中的一部分。

更通俗地说，也就是 Nginx 的目的是为了解决 C10k 问题。这个问题指明了一个共同的评论点，根据当前的计算机技术状态和网络扩展性仅允许一台计算机（从主流产业）连接 10 000（同时并发的网络连接），受到操作系统和软件的限制。由于技术的进步，现在该值已经不再有代表性了，但在那个时候，这个问题却被认为是非常严重的，因此引发了较大的 Web 服务开发，例如 Lighttpd、Cherokee，以及 Nginx。

10.6 从 Nginx 接收客户端请求处理的角度来说，它与 Apache 有何不同

这一个问题问到了问题的根本，Nginx 采用的是事件驱动结构，使用异步套接字来接收请求，是一种非阻塞结构，不使用单独的线程处理，目的是为了减少内存和 CPU 的开销。而 Apache 采用同步套接字、线程和进程，每一个请求都是一个单独的进程或线程。

10.7 安装完成 Nginx 后，如何查看 Nginx 的版本

找到 Nginx 命令的绝对路径，然后执行：nginx -v。

例如：

```
[root@backup dk]# /usr/local/nginx0.8.53/sbin/nginx -v
nginx version: nginx/0.8.53
```

10.8 安装完成 Nginx 后，如何查看 configure 时的配置

找到 Nginx 命令的绝对路径，然后执行：nginx -V。

例如：

```
[root@mail nginx-0.8.53]# /usr/local/nginx0.8.53/sbin/nginx -V
nginx version: nginx/0.8.53
built by gcc 4.1.2 20070626 (Red Hat 4.1.2-14)
configure arguments: --prefix=/usr/local/nginx0.8.53/ --add-module=/root
/nginx-accesskey-2.0.3
```

10.9 启动 Nginx 后，能不能看到 Nginx 当前都支持哪些模块

直接是看不出来的，不像 Apache 一样，可以看到动态载入的模块，但是你可以使用“nginx -V”查看都禁止哪些模块，又添加了哪些模块，就能够确定现在使用的模块了。

10.10 Https 仅能用在指定的目录下吗

这是一个来自于forum.nginx.org的问题，提问者是这么写的，他想在一个指定的目录location下使用 SSL，配置如下：

```
server {
 listen   10.30.1.50:80 default_server backlog=1024 rcvbuf=32k sndbuf=8k;
 listen   10.30.1.50:443 ssl;
 server name  www.domain.com;
 ssl certificate  domain.com.crt;
 ssl certificate key domain.com.key;
…
 location / {
     try_files $uri $uri/ /index.php?$uri&$args;
 }

 location /dir/ {
     auth basic  "Restricted Access";
     auth basic user file htpasswd;
     rewrite ^ https://www.domain.com/dir$request uri? permanent;
 }
…
}
```

换句话说，如果你想通过 HTTP 值（这里指的是 HTTP 协议中变量的值，例如$host、$port 等），然后再利用$scheme 重定向到 HTTPS（在这里并不想使用任何 if 条件），如何能够做到？

Igor Sysoev 的回复如下：

```
location /dir/ {
if ($scheme != https) {
rewrite ^ https://www.domain.com/dir$request_uri? permanent;
}
}
```

但是并不推荐这么使用，最好是使用单独的 Server。

因此，最好还是采用以下这种形式：

```
server {
   listen 443;
```

```
    ssl on;
    ssl_certificate  domain.com.crt;
    ssl_certificate_key  domain.com.key;
    server name   www.domain.com;
    index index.html index.htm index.php;
    root /var/www/test;
}
```

第 2 部分

Nginx 服务器的功能模块

我们将在本部分来认识 Nginx 的功能模块。这部分讲述的功能模块都是用于增强 Nginx 服务器的功能的，而有关 Nginx 与应用服务器相配合、实现动静分离的模块，例如代理模块、FastCGI 模块、Uwscgi 模块等，都将在卷 2 中讲述。

第 11 章　限 制 流 量

对于提供下载的网站，肯定是要进行流量控制的，例如 BBS、视频服务，还有其他专门提供下载的网站。在 Nginx 中我们完全可以做到限流，由 Core 模块提供了 limit_rate、limit_rate_after 指令。

11.1 指令

通过以下两条指令来完成限制流量。

指令名称：limit_rate

语法：limit_rate speed

默认值：no

使用环境：http，server，location，if in location

功能：该指令用于指定向客户端传输数据的速度，速度的单位是每秒传输的字节数。需要明白的一点是，该限制只是针对一个连接的设定，也就是说，如果同时有两个连接，那么它的速度将会是该指令设置值的两倍。

如果需要在 Server 级别对某些客户端限制速度，对于这种情况——这个指令可能并不适合，但是可设置$limit_rate 变量，为该变量传递相应的值来实现，例如：

```
server {
  if ($slow) {
set $limit_rate  4k;
  }
}
```

当然也可以通过设置 X-Accel-Limit-Rate 头（来自于 NginxXSendfile 模块）来控制由 proxy_pass（来自于 HttpProxyModule 模块）返回的相应数据的速率，而不使用 X-Accel-Redirect 头。

指令名称：limit_rate_after

语法：limit_rate_after size

默认值：limit_rate_after 1m

使用环境：http，server，location，if in location 中的 if 字段

功能：该指令中的“after”提示了我们，可以这样理解“在……后再限制速率为……”，它的语法为：limit_rate_after time（这是官方微博 http://wiki.nginx.org/HttpCoreModule#limit_rate 上的语法），它的意思是以最大的速度下载 time 时长后。但在实际的使用中发现指令 limit_rate_after 的参数是一个下载字节量的大小值，而不是时间值，因此命令“limit_rate_after 3m”解释为：以最大的速度下载 3MB 后。

11.2 实例配置

看下面的配置(这是一个视频服务器上的配置片段),通过两条命令限制了访问者的下载速度:

```
    location /download {
    limit rate after 3m;
    limit rate 512k;
  }
```

我们分析一下这两条命令:

limit_rate,相对于 limit_rate_after 命令,这个命令已经开始限速了,它的语法为:limit_rate speed,表示限制为的速率。该命令可以用在 HTTP,Server,Location 以及 Location 中的 if 区段,没有默认值。

有了对这两条命令的了解,我们便可以解释以上配置文件中的意思:当一个客户端连接后,将以最快的速度下载 3MB(3 兆的数据量),然后再以 51KB 的速度下载。

我们测试一下:

```
[root@ download ~]# wget http://192.168.3.133/download/873 6.flv
--15:03:06--  http://192.168.3.133/download/873 6.flv
   => '873 6.flv '
Connecting to 192.168.3.133:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10,673,892 [application/zip]

65%[=======================> ] 10,673,892  511.81K/s   ETA 00:00

15:03:24 (571.93 KB/s) - '873_6.flv ' saved [10,673,892/10,673,892]
```

在刚开始的 3MB 中很快,然后速度就慢下来了。

第 12 章　限制用户并发连接数

可以通过 l imit zone 模块来达到限制用户的连接数的目的，即限制同一用户 IP 地址的并发连接数。

该模块提供了两个命令：limit_zone 和 limit_conn，其中，limit_zone 只能用在 http 区段，而 limit_conn 可以用在 http、server、location 区段。

12.1 示例配置

```
http {
  limit zone   one  $binary remote addr  10m;

  server {
location /download/ {
  limit_conn   one  1;
}
  }
}
```

12.2 指令

limit_zone 模块提供了以下 3 条指令。

指令名称：limit_zone

语法：limit_zone zone_name $variable memory_max_size

默认值：no

使用环境：http

功能：该指令用于定义一个 zone，该 zome 将会被用于存储会话的状态。能够存储的会话数量是由分配交付的变量和 memory_max_size 的大小决定的。

例如：

```
limit_zone one $binary_remote_addr 10m;
```

客户端的 IP 地址被用作会话，**注意，这里使用的是$binary_remote_addr** 而不是 **$remote_addr**，这是因为，$remote_addr 的长度为 7～15 个字节，它的会话信息的长度为 32 或 64 bytes；$binary_remote_addr 的长度为 4 字节，会话信息的长度为 32 字节。当设置 1MB 的一个 zone 时，如果使用$binary_remote_addr 方式，该 zone 将会存放 32000 个会话。

指令名称：limit_conn

语法：limit_conn zone_name max_clients_per_ip

默认值：no

使用环境：http，server，location

功能：该指令用于为一个会话设定最大的并发连接数。如果并发请求数超过这个限制，那么将会出现"Service unavailable"（503）。

例如：

```
limit_zone  one  $binary_remote_addr  10m;

server {
  location /download/ {
  limit_conn   one  1;
}
```

这个设置将会使得来自于同一个 IP 的并发连接不能超过 1 个。

指令名称：limit_conn_log_level

语法：limit_conn_log_level info | notice | warn | error

默认值：error

使用环境： http，server，location

功能：该指令用于设置日志的错误级别，当达到连接限制时，将会产生错误日志。

12.3 使用实例

看下面的一个例子：

```
[root@flv conf]# cat nginx.conf

worker_processes  4;

events {
worker_connections  10240;
}

http {
include   mime.types;
default_type  application/octet-stream;
sendfile   on;
keepalive_timeout  65;
limit_zone   flv_down  $binary_remote_addr  10m;

server {
listen   80;
```

```
    server_name  flv.xxx.com;
    …
    location /download {
       limit_conn   flv_down  1;
    }

    error page   500 502 503 504  /50x.html;
    location = /50x.html {
    root   html;
    }
    }
```

在这个例子中，我们使用了两个命令：limit_zone 和 limit_conn。

通过以下两步来完成。

（1）使用 limit_zone 命令定义一个 zone：

```
limit_zone flv_down $binary_remote_addr 10m;
```

在这里，命令 limit_zone 有三个参数。

- flv_down：是一个 zone 名字，它只是随便定义的一个名字（最好定义一个有意义的名字，因为下一条命令中要使用到，再者如果定义的 zone 名字多了，连自己都不知道是干什么的了）；
- $binary_remote_addr：是一个变量，用来做客户端之间的区别，也可以选择其他的变量来作为区别，但是最典型的还是$binary_remote_addr（这里选用的是二进制格式的 IP 地址，会比 ASCII 格式更高效）；
- 10m：用于限制大小，memory_max_size 用于设定分配给存储会话状态表的最大值。

（2）使用 limit_conn 命令来实现 limit_zone 命令定义的 zone：

```
limit_conn  flv_down  1;
```

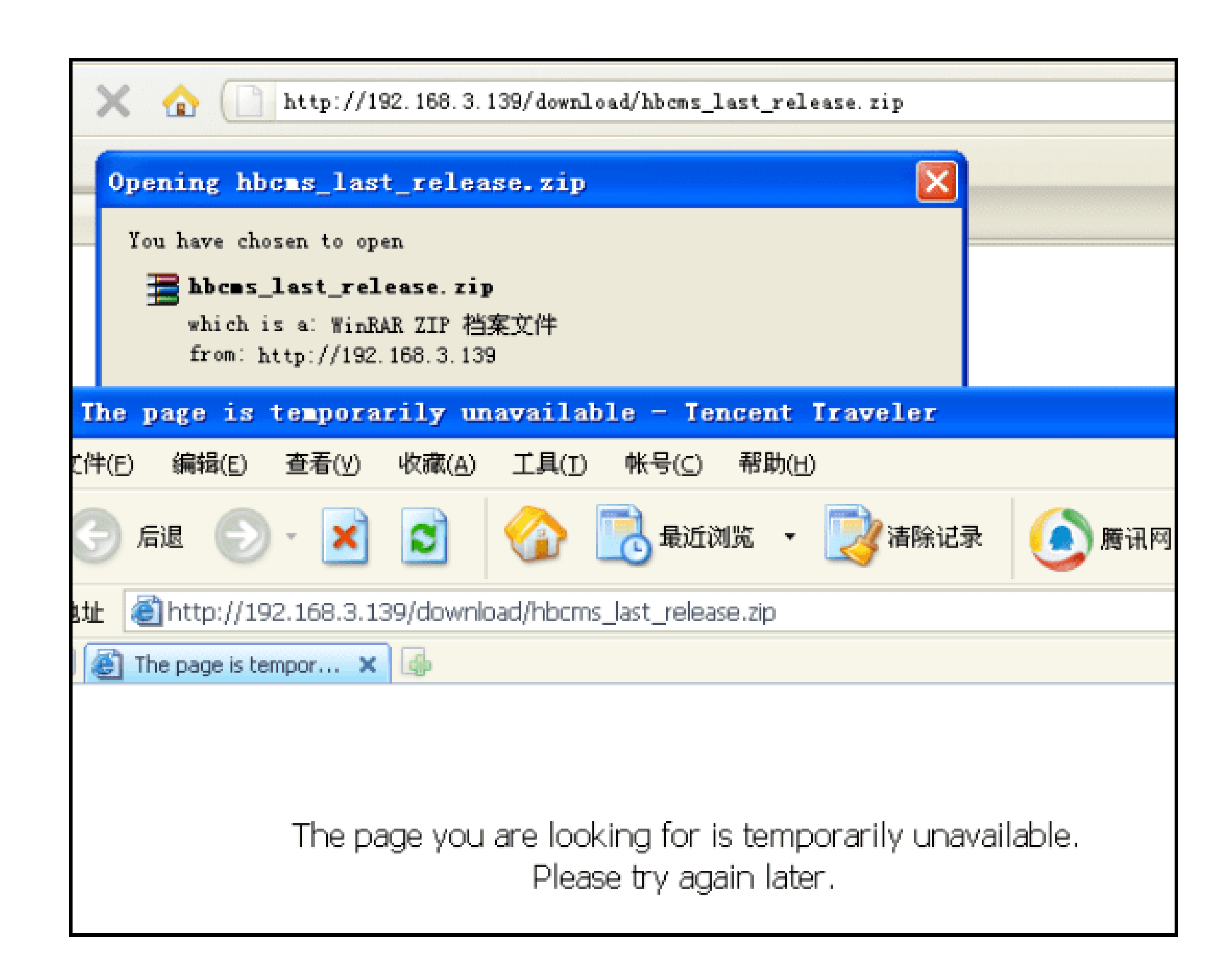
在这里，命令 limit_conn 有两个参数。

- flv_down：是一个 zone 名字，这是由第一个命令定义的；
- 1：它定义的是同时连接的数量。

limit_zone 和 limit_conn 这两条命令的结果就是使得共享同一个$binary_remote_addr 的请求受到连接的限制（同时只能一个连接）。如果达到限制，那么所有其他的连接请求将会被“503 Service unavailable HTTP response”回答（就是说服务无效）。

重新载入配置文件后再访问。我们通过两个浏览器来访问这个资源，首先访问的那个浏览器会显示下载另存为，而后的那个浏览器就不会是这种情况了，相反，它会显示“The page you are looking for is temporarily unavailable.Please try again later.”这个提示就是说，服务器太忙，暂时无法提供服务——实际上是同一 IP 连接受限制了。

下面是截图：



> 在上面的配置文件中有一点需要注意，那就是客户端 IP：客户端的 IP 地址被用作会话，还有一点是这里使用了变量$binary_remote_addr 来表示 IP 地址，而不是变量$remote_addr。

为什么这么做呢？ $remote_addr 的长度为 7～15 个字节，它的会话信息的长度为 32 或 64 bytes；而 $binary_remote_addr 的长度值总是 4 个字节，会话信息的长度总是 32 个字节。因此，二进制格式的 IP 地址比 ASCII 更高效。如果把 zone 的大小设置为 1MB，那么可能存储的会话约为 32000（1MB/32=32768）。

第 13 章　修改或隐藏 Nginx 的版本号

由于各种原因，可能需要修改或是隐藏 Nginx 的版本号，为了 Nginx 的安全性——对于一个老的 Nginx 版本，你可能不再打算对它升级了，但是出于对安全因素的考虑，因此想将其版本号隐藏；或者是吃饱了撑的没事干（开个玩笑，我知道你很忙！）打算将 Nginx 的名字及版本号全部替换掉，这样看上去貌似你自己开发的 Web 服务器，这部分内容就解决这两个问题。尤其对于第二个问题一定要改得有水平。

13.1 隐藏版本号

隐藏 Nginx 的版本号很简单，Nginx 的 HttpCoreModule 提供了一条 server_tokens 指令，这里将该条指令设置为“server_tokens off”就可以了。

首先访问一下，查看现有的版本：

```
[root@ mail ~]# curl --head http://192.168.3.139
HTTP/1.1 403 Forbidden
Server: nginx/0.8.53
Date: Thu, 09 Dec 2010 00:02:04 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
```

通过访问，本人得到的是“**Server: nginx/0.8.53**”。

然后在配置文件的 http 区段中插入“**server_tokens off;**”，重新载入配置文件：

```
[root@mail ~]# vi /usr/local/nginx0.8/conf/nginx.conf

worker processes  1;

events {
worker_connections  1024;
}

http {
include   mime.types;
default type  application/octet-stream;
server_tokens  off;
expires 5s;
sendfileon;

keepalive_timeout  65;
```

```
include "sites-enabled/mail*";

}
[root@mail ~]#service nginx reload
```

再次访问，看下图，Nginx 没有了版本号。



13.2 修改版本号

修改版本号的方法比隐藏版本号的方法要复杂，它需要在配置安装 Nginx 之前就进行。下载完成 Nginx 并解压后，首先对源码进行修改。源码文件都在二级目录"nginx-0.8.53/src/"下，找到文件"src/core/nginx.h"，然后按照下面的代码中指出的两行（已用粗体标明）。对其进行修改。

```
[root@mail nginx-0.8.53]# vi src/core/nginx.h

…

#define nginx version 8053
#define NGINX_VERSION  "10.0"
#define NGINX_VER  "jh/" NGINX_VERSION

#define NGINX_VAR  "NGINX"
#define NGX_OLDPID_EXT ".oldbin"


#endif /* _NGINX_H_INCLUDED_ */
```

对这两行，你可以随便填写。如果是为了迷惑别人，你可以修改为 Apache 3.0 或是 Apache 2.0，或者是微软的 IIS（估计这是在自找麻烦！）以及其他的文本服务器名称或版本号都行；如果是想牛 X 一下，那么就自己起一个响儿不亮的名字吧！修改完成后就可进行编译安装了。安装完成后访问一下，看看效果：



没错，已经按照我们的意思被修改了。

第 14 章　配置 FLV 服务器

配置 FLV 服务器，我们可以通过 ngx_http_flv_module 模块来实现，但需要注意的是，该模块仅提供了 FLV 访问的必要条件，但没有提供，好像是也不能提供关键帧数据的制作，对于播放器，也需要我们自己找或者是开发。

配置 FLV 服务器，需要 Nginx 的 HttpFlvStreamModule 模块，该模块支持对 FLV（Flash 格式 S）文件的拖动播放。

模块 ngx_http_flv_module 提供了特殊的配置方法：

- 为客户端请求的文件添加 FLV 头；
- 从指定的位置传输文件，在请求中使用了 start=XXX 参数。

在默认安装的情况下，该模块没有包含在内，因此如果想使用该模块，则需要在编译安装时指定--with-http_flv_module 参数，以便将该模块编译安装。

14.1 示例配置

```
location ~ \.flv$ {
  flv;
}
```

14.2 指令

该模块只提供了一条指令，那就是 flv。

指令名称：flv

语法：flv

默认值：None

使用环境：location

功能：在相应的 location 中开启 FLV 支持。

14.3 使用实例

要配置一个可用 FLV，需要通过以下步骤。

1. 安装具有 FLV 功能的 Nginx

例如，以下安装方式：

```
[root@flv nginx-1.0.2]# ./configure --prefix=/usr/local/nginx-1.0.2-flv  \
> --with-http_flv_module
```

```
[root@flv nginx-1.0.2]#make
[root@flv nginx-1.0.2]#make install
```

添加配置：

```
server {
  listen   80;
  server name  192.168.1.105;
  roothtml;
  limit_rate_after 5m;
  limit_rate 512k;
index   index.html;
  location ~ \.flv$ {
 root /var/www/flv;
   flv;
}
location ~ \.mp4$ {
 root /var/www/mp4;
 flv;
  }


 }
```

2. yamdi

该软件的全名叫 Yet Another MetaData Injector，其功能很强大，支持很多操作系统，对于 Windows 还支持 64 位系统，并且还支持 H.264。对于本软件就不多分析了，够我们用就可以了。

下载并安装 yamdi

```
[root@flv ~]# wget http://cdnetworks-kr-2.dl.sourceforge.net/  \
> project/yamdi/yamdi/1.8/yamdi-1.8.tar.gz
[root@flv ~]# tar -zxvf yamdi-1.8.tar.gz
yamdi-1.8
yamdi-1.8/CHANGES
yamdi-1.8/LICENSE
yamdi-1.8/Makefile
yamdi-1.8/Makefile.mingw32
yamdi-1.8/README
yamdi-1.8/man1
yamdi-1.8/yamdi.c
yamdi-1.8/man1/yamdi.1
[root@ flv ~]# cd yamdi-1.8
[root@ flv yamdi-1.8]# make
gcc -O2 -Wall yamdi.c -o yamdi
[root@ flv yamdi-1.8]# make install
install -m 0755 -o root yamdi /usr/local/bin
```

需要注意的是，命令 yamdi 被安装在/usr/local/bin 目录下，有关命令 yamdi 的详细用法，

可以参考源代码中提供的 man 文档。

使用 yamdi

这里我们准备一个文件，以便后面使用：

```
[root@flv html]# yamdi -i 62664.flv -o 7345.flv
```

简单地说一下，-i 表示输入文件。在这里输入文件为 62664.flv，即它是没有添加过关键帧的文件；-o 表示输出文件，在这里是 7345.flv，它是添加过关键帧的。在对这两个文件的访问中会发现，播放 62664.flv 是不能实现拖动操作的，而 7345.flv 则可以。

3. 下载并设置 JW player

JW player 是一个开源的 FLV 播放器，支持 MP4。

```
    [root@flv ~]# wget http://www.longtailvideo.com/jw/upload/mediaplayer
viral.zip
    [root@flv ~]# unzip mediaplayer-viral.zip
    Archive:  mediaplayer-viral.zip
      creating: mediaplayer-5.7-viral/
      inflating: mediaplayer-5.7-viral/JW Player Quick Start Guide.pdf
      inflating: mediaplayer-5.7-viral/jwplayer.js
      inflating: mediaplayer-5.7-viral/license.txt
      inflating: mediaplayer-5.7-viral/player.swf
      inflating: mediaplayer-5.7-viral/preview.jpg
      inflating: mediaplayer-5.7-viral/readme.html
      inflating: mediaplayer-5.7-viral/swfobject.js
      inflating: mediaplayer-5.7-viral/video.mp4
```

注意在解压包中，**player.swf** 和 **jwplayer.js** 是我们需要的文件，将其复制到 Nginx 的 Web 目录下：

```
    [root@flv ~]#cd mediaplayer-5.7-viral/
    [root@mailmediaplayer-5.7-viral]#cpjwplayer.jsplayer.swf  /usr/local/
  nginx-1.0.2-flv/html/
```

到现在为止，FLV 服务器已经架设完毕。

我们访问测试一下。

访问方法：

http://flv.xx.com/player.swf?type=http&file=7345.flv

访问协议：↑FLV 服务器地址↑播放器名称↑http 分发方式↑访问的文件名

结果如右图所示。



另外，参阅 **JW Player Quick Start Guide.pdf** 文档，可以将播放器嵌入在网页中，还可以设定画面的大小，播放列表等，但这些不是我们的工作，它属于 HTML 代码或者是美编处理部分了。

4. 另类实现方法

还有一种方法是通过 nginx_mod_h264_streaming 实现的，我们来看一下。

下载 nginx_mod_h264_streaming 模块

```
[root@flv ~]wget http://h264.code-shop.com/download/nginx mod h264
streaming-2.2.7.tar.gz
[root@flv ~]# tar -zxvf nginx_mod_h264_streaming-2.2.7.tar.gz
```

修改 Makefile

可能需要修改 Makefile 文件，根据实际情况修改：

```
[root@mail nginx mod h264 streaming-2.2.7]# vi Makefile

# vim:noexpandtab:sw=2 ts=2

.PHONY: all dist install clean

HOME=$ (shell echo ~)
PWD=$ (shell pwd)
#NGINX=$ (HOME) /nginx-0.8.29/
#NGINX=$ (HOME) /nginx-0.7.33/
 NGINX=/root/nginx-1.0.2/

VERSION='./version.shlimit_conn
DIST_NAME=nginx_mod_h264_streaming-$ (VERSION)

all:
cd $ (NGINX)  && ./configure
   --sbin-path=/usr/local/sbin
   --add-module=/root/nginx_mod_h264_streaming-2.2.7
   --with-debug --with-http_flv_module
make --directory=$ (NGINX)

…//省略
```

注意黑体字部分，它使用了 HttpFlvStreamModule 模块，即**--with-http_flv_module** 参数。

编译安装

```
[root@flv nginx-1.0.2]# ./configure
   --prefix=/usr/local/nginx-1.0.2-h264-flv
   --add-module=/root/nginx mod h264 streaming-2.2.7
[root@flv nginx-1.0.2]#make
[root@flv nginx-1.0.2]#make install
```

如果在 configure 过程中出现以下错误：

```
/root/nginx_mod_h264_streaming-2.2.7/src/ngx_http_streaming_module.c:
In function 'ngx_streaming_handler':
```

```
/root/nginx_mod_h264_streaming-2.2.7/src/ngx_http_streaming_module.c:15
8: error: 'ngx_http_request_t' has no member named 'zero_in_uri'
make[1]: *** [objs/addon/src/ngx_http_h264_streaming_module.o] Error 1
make[1]: Leaving directory `/root/nginx-0.8.54'
make: *** [build] Error 2
```

将 src/ngx_http_streaming_module.c 文件中以下代码删除或者注释掉就可以了：

```
/* TODO: Win32 */
if (r->zero_in_uri)
{
return NGX_DECLINED;
}
```

如果你没有对该文件做过更改，则应该在源码的第 157～161 行。此问题是由于版本原因引起，在此不再讨论。

修改完之后，记得先执行 make clean，然后再重新执行 configure、make，最后 make install。

nginx_mod_h264_streaming 模块的用法

我们再看一下 src/ngx_http_streaming_module.c 文件，看以下部分：

```
[root@flv nginx mod h264 streaming-2.2.7]# vi  src/ngx http streaming
module.c


#if 0
/* Mod-H264-Streaming configuration

server {
  listen 82;
  server_name  localhost;

  location ~ \.mp4$ {
root /var/www;
mp4;
  }
}

*/

/* Mod-Smooth-Streaming configuration

server {
  listen 82;
  server name localhost;
```

```
    rewrite ^(.*/)?(.*)\.([is])sm/[Mm]anifest$ $1$2.$3sm/$2.ismc last;
    rewrite ^(.*/)?(.*)\.([is])sm/QualityLevels\(([0-9]+)\)/Fragments\
((.*)=([0-9]+)\) (.*)$ $1$2.$3sm/$2.ism?bitrate=$4&$5=$6 last;

    location ~ \.ism$ {
   root /var/www;
   ism;
    }
   }
   */
   #endif
```

添加配置

在这个源码文件中嵌入了该模块的用法，注意黑体字部分，因此我们的配置文件应该这么写：

```
   server {
    listen   80;
    server_name  192.168.1.105;
    roothtml;
    limit_rate_after 5m;
    limit_rate 512k;
  index   index.html;
    location ~ \.flv$ {
   root /var/www/flv;
      mp4;
  }
  location ~ \.mp4$ {
   root /var/www/mp4;
   mp4;
    }

   }
```

5. 访问测试

启动 Nginx，访问 http://flv.xx.com/player.swf?type=http&file=4315.mp4，如右图所示。



如果我们在执行 configure 时使用了--with-http_flv_module 选项，例如：

```
[root@flv nginx-1.0.2]# ./configure  --prefix=/usr/local/nginx-1.0.2
-mp4-flv
  --add-module=/root/nginx_mod_h264_streaming-2.2.7
  --with-http_flv_module
```

那么在 Nginx 的配置文件中可以这么配置：

```
    location ~ \.flv$ {
  root /var/www/flv;
      flv;
 }
 location ~ \.mp4$ {
  root /var/www/mp4;
  mp4;
    }
```

第 15 章　Nginx 的访问控制

Nginx 提供了一个 HttpAccessModule 模块，用于 Nginx 实现基于网络层的控制。它类似 Linux 中的/etc/hosts.allow 和/etc/hosts.deny，也就是 tcpd wrapper 机制。Apache 同样支持这类访问控制。

nginx_http_access_module 模块能够针对指定 IP 地址客户端地址进行控制，该模块于 0.8.22 版本提供。从 0.8.22 版本之后的 Nginx 开始支持 IPv 6。

nginx_http_access_module 模块是通过检查来访问客户端的 IP 是否通过匹配访问规则的方式，匹配 allow 规则，则可以进行访问；否则就是 deny。

> 需要注意的一点是，它的检测顺序，就是按照在配置文件中的配置顺序来顺序执行，匹配首条的规则将会被使用，因此，要注意在配置文件中配置的顺序。

15.1 示例配置

```
location / {
  deny192.168.1.1;
  allow   192.168.1.0/24;
  allow   10.1.1.0/16;
  allow   2620:100:e000::8001;
  denyall;
}
```

15.2 指令

该模块提供了两条指令，即 allow 和 deny。

指令名称：allow

语法：allow [address | CIDR | all]

默认值：none

使用环境：http，server，location，limit_except

功能：由该指令指定的网络地址或者是 IP 地址允许访问。

指令名称：deny

语法：deny [address | CIDR | all]

默认值：none

使用环境：http，server，location，limit_except

功能：由该指令指定的网络地址或者是 IP 地址禁止访问。

15.3 使用实例

看下面的配置。在这个配置中，我们使用了 4 个 location：

```
location /c1/ {
  allow   192.168.3.0/24;
  allow   100.100.0.0/16;
  deny all;
  index  index.html index.htm;
}

location /c2/ {
  deny    192.168.3.248;
  allow   192.168.3.0/24;
  allow   100.100.0.0/16;
  deny all;
  index  index.html index.htm;
}

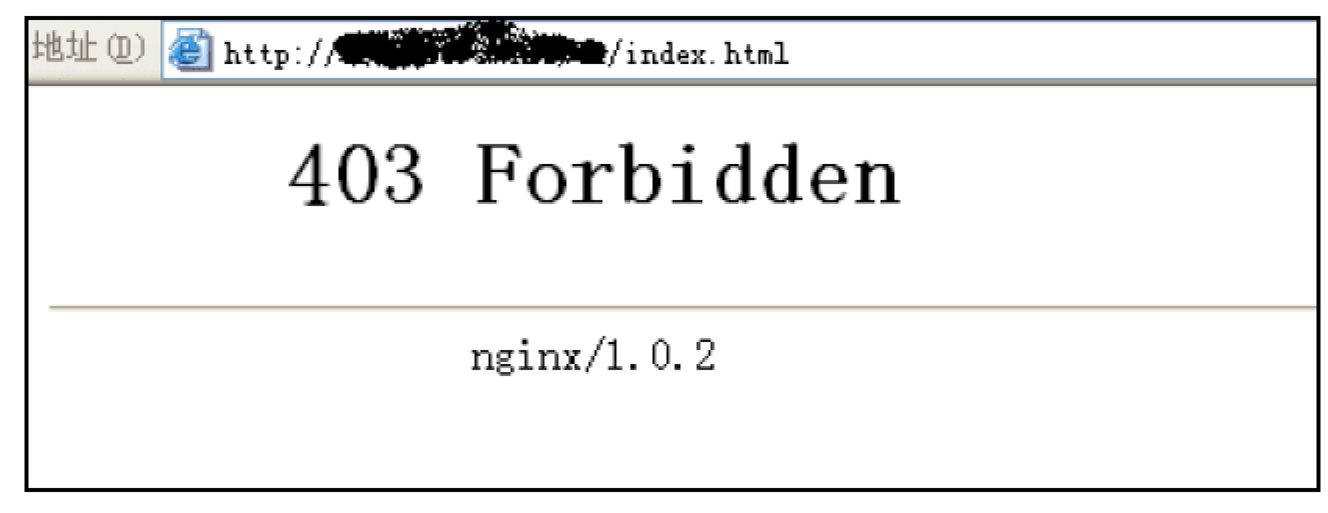
location /c3/ {
  allow   192.168.3.0/24;
  deny    192.168.3.248;
  allow   100.100.0.0/16;
  deny all;
  index  index.html index.htm;
}

location /c4/ {
  deny    192.168.3.248;
  allow   192.168.3.0/24;
  allow   100.100.0.0/16;
  deny all;
  index  index.html index.htm;
  error_page  403  http:// www.xx.com /ok.html;
}
```

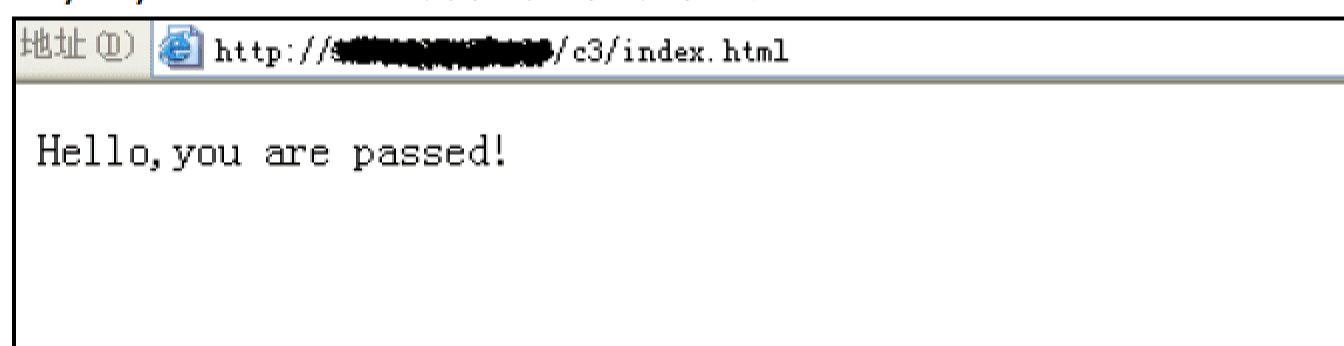
下面我们对每一个 location 进行分析。

在第一个 location 中，遵循了先 allow 后 deny 的原则，因此允许的两个网段，即 192.168.3.0/24 和 100.100.0.0/16 都没问题，其他的将会被 deny。

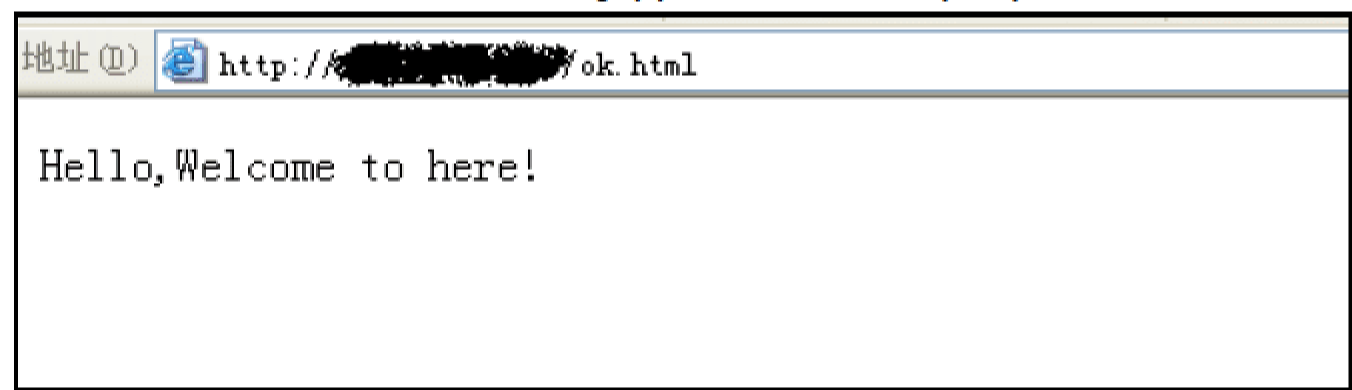
在第二个 location 中，我们先 deny 了 192.168.3.248 这个地址，接着又允许了网段 192.168.3.0/24 的访问，遵循第一个匹配的规则优先。因此，IP 为 192.168.3.248 的地址是无法访问的，例如，访问 http://www.xx.com/c2/index.html 的结果将是以下页面：



在第三个 location 中，我们先 allow 了 192.168.3.0/24 网段，然后又 deny 了该网段的一个 IP 地址——192.168.3.248，同样遵循第一个匹配的规则优先，如果 IP 地址为 192.168.3.248 的客户端来访问，那么首先被 allow 192.168.3.0/24 规则匹配通过。因此，在我们访问 http://192.168.3.139/c3/index.html 时得到的页面如下：



在第四个 location 中，唯一与第二个 location 不同的是多添加了一条 error_page 403 http://www.xx.com /ok.html 指令，它的作用很简单，就是针对 403 错误设计的，如果发生 403 错误，那么就重定向到其他地方。我们看一下访问 http://www.xx.com/c4/index.htm 产生的页面：



没错。我们访问的是 http://www.xx.com/c4/index.htm，而页面被转到 http://www.xx.com/ok.html，这个要比返回“403 Forbidden”好多了，还可以做其他的使用。

第 16 章　提供 FTP 下载

在 Nginx 中提供了一个 HttpAutoindexModule 模块，它的功能是在一个没有 index.html 的目录中提供文件的自动列表。因此，我们可以使用它来实现 FTP 下载功能。需要明白的一点是，只有 ngx_http_index_module 找不到 index 文件时，客户端的请求才会到达 ngx_http_autoindex_module。

16.1 示例配置

```
location  /  {
  autoindex  on;
  autoindex_exact_size off;
  autoindex_localtime on;
}
```

其实对于模拟提供 FTP 下载，有这个配置足够了。但是该模块还提供了其他指令，因此在这里我们也了解一下。

16.2 指令

HttpAutoindexModule 模块提供了三条指令，便于我们更好地使用。

指令名称：autoindex

语法：autoindex [on | off]

默认值：autoindex off

使用环境：http，server，location

功能：启用或者禁用自动目录列表。

指令名称：autoindex_exact_size

语法：autoindex_exact_size [on | off]

默认值：autoindex_exact_size on

使用环境：http，server，location

功能：该指令用于在目录列表中设定文件大小的格式，如果是以精确的大小显示，那么使用 KB，如果是以取整表示，那么使用 KB、MB 或者 GB，默认为精确显示大小。

指令名称：autoindex_localtime

语法：autoindex_localtime [on | off]

默认值：autoindex_localtime off

使用环境：http，server，location

功能：是否在目录列表中以本地时间显示文件的时间，默认为 off，即使用 GMT 时间。使用

该指令要注意，在时间显示上有很大不同。

16.3 使用实例

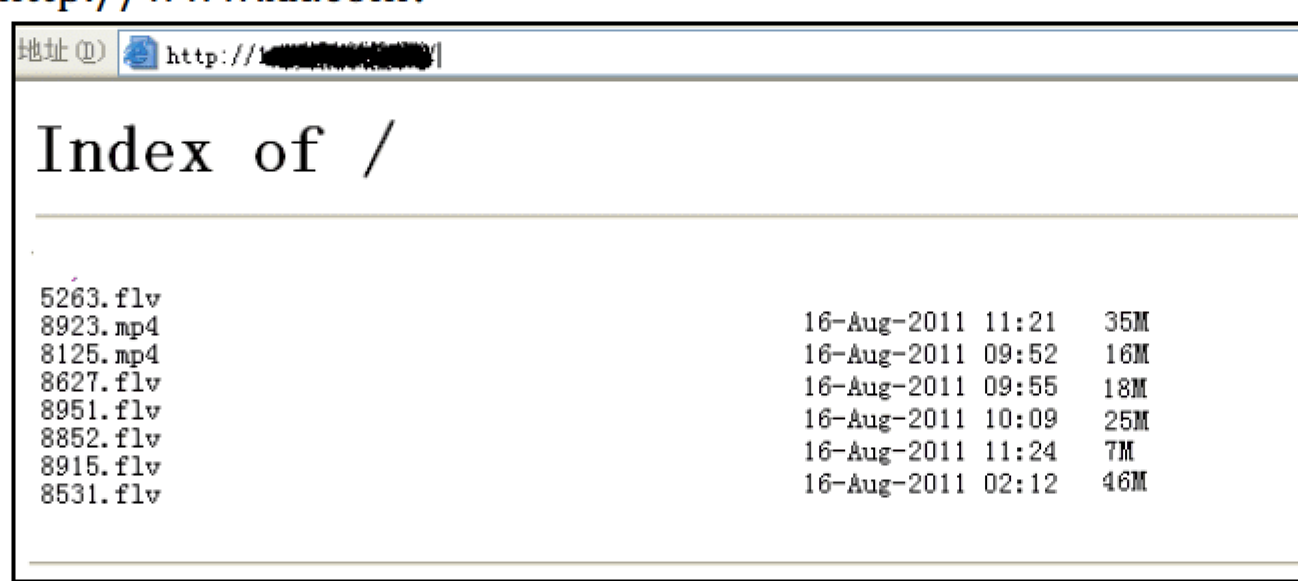
我们先看以下这个配置：

```
location / {
  root   html;
  autoindex  on;
  autoindex_exact_size off;
  limit_rate 1k;
  index  index.html index.htm;
}
```

在这个配置中，我们添加了三条指令，其中前两条和自动目录列表有关，而第三条则是用于控制下载速度，在这里设置为了 1k，纯粹是为了查看效果，在具体的应用中要按照具体的网络速度来配置。

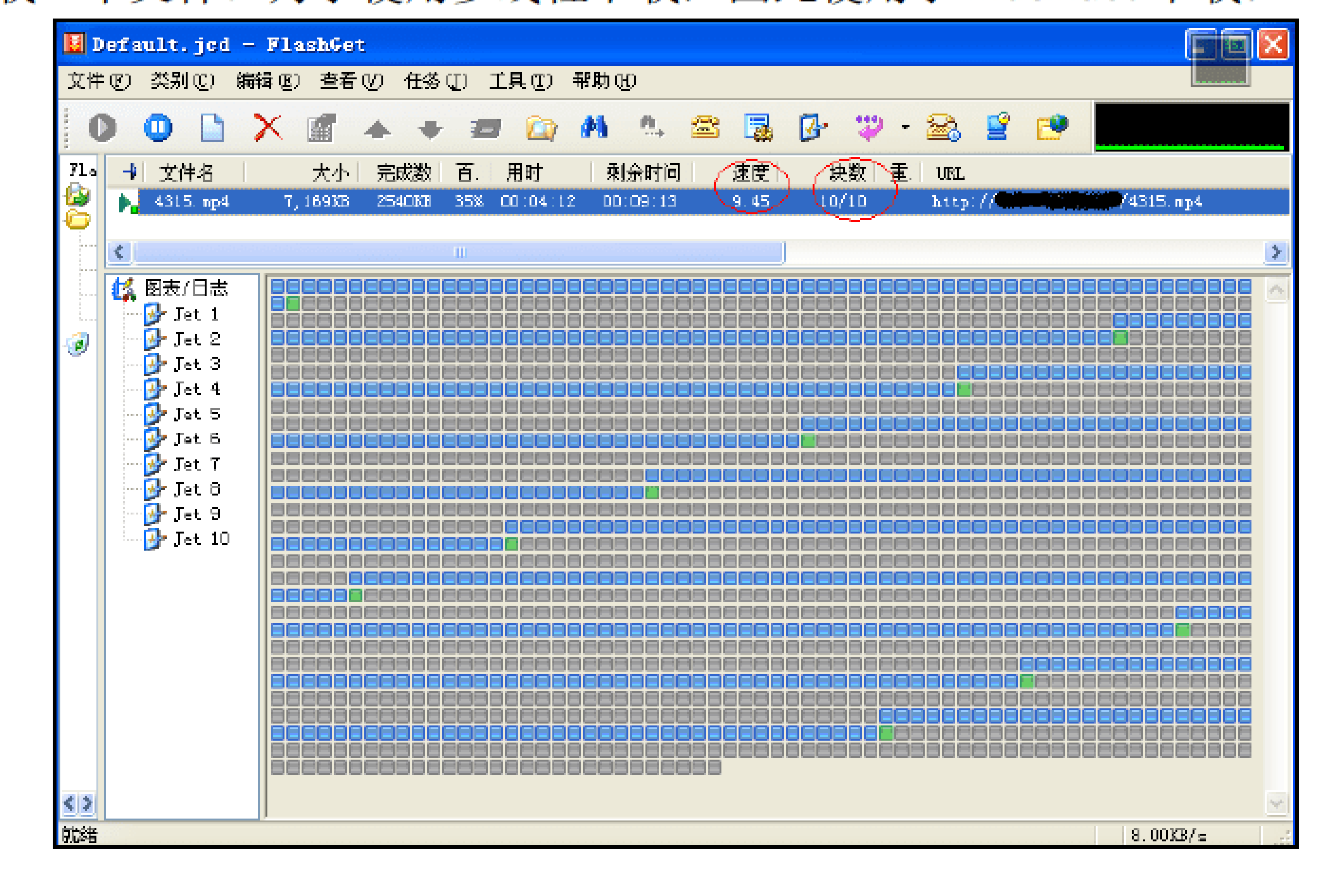
访问测试

下面我们访问 http://www.xx.com：



可见由于该目录没有 index 文件，因此就以自动列表的形式显示了。

下面我们下载一个文件。为了使用多线程下载，因此使用了 FlashGet 下载：



注意图中的数据，第一处，速度为 9.45，第二处为 10/10，第三处就是左边列的从 Jet1～Jet10，这说明我们使用的是 10 个线程下载的，因此就分成了 10 个块。在前面的配置中，我们将 limit_rate 设置为了 1k，因此 10 个线程最大也就是 10k，因此符合配置。然后我们再去服务器端看一下连接数：

```
[root@mail html]# lsof -i:80|grep 192.168.3.248|wc -l
10
```

很明显，来自于 192.168.3.248 客户端的连接数为 10 个。

因此，如果是用 Nginx 提供 FTP 下载，那么这些数据都是要计算的。无论是 Nginx 的最大连接数还是网络的带宽，都不能超出。

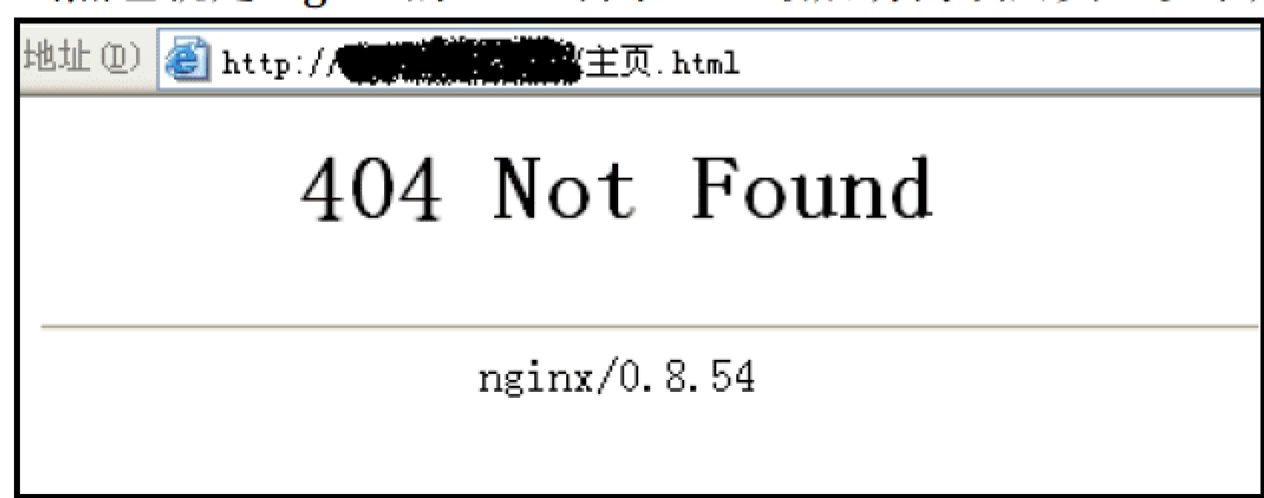
第 17 章　Nginx 与编码

提到编码，可能每个人都会有很多感慨，操作系统有编码、Nginx 服务器有编码，网页也有编码。对于网页的编码而言，还有文件名本身的编码和文件内容的编码之分。

为了更好地理解这些关系，我们先来了解一下文件和文件名的编码。

17.1　文件和文件名的编码

假设我们在 Windows 系统下创建一个.html 文件，文件名为“主页.html”，然后将其传入操作系统下的目录（当然也就是 Nginx 的 Web 目录），最后访问该网页，以下是访问的结果：



我们确实是将该文件传到了相应的目录下，而为什么是这种结果呢？答案就在于编码！我们看看被传到 Linux 的文件名称：

```
[root@mfsmaster html]# ls
???.html
```

> 这里需要说明的一点是，浏览器在默认情况下，URI 的编码是以 UTF-8 的方式编码后向服务器发送的，因此在 URL 中出现中文同样会以 UTF-8 的方式编码后发送到服务器，如果系统的字符集、文件名的字符集和 URI 的字符集不一致就会出现解码问题。

文件名称已成乱码，因此，无法找到文件名，最终出现 404 错误。造成这一结果是因为传入文件的文件名称编码与现在 Linux 系统的编码不一致所造成的（当然首先要让你的登录工具，例如，SecureCRT，与当前系统的编码一致），Linux 系统的编码我们可以查看一下：

```
[root@mfsmaster ~]# echo $LANG
zh_CN.utf-8
```

而我们的网页文件是从 Windows 系统传入的，因为 Windows 系统的默认编码为 GBK，因此需要将编码转换后才可以使用。

GBK：汉字国标扩展码,基本上采用了原来 GB2312-80 所有的汉字及码位，并涵盖了原 Unicode 中所有的汉字 20902 个，总共收录了 883 个符号，21003 个汉字及提供了 1894 个造字码位。Microsoft 简体中文版 Windows 95 就是以 GBK 为内码,又由于 GBK 同时也涵盖了 Unicode 所有 CJK 汉字，所以也可以和 Unicode 做一一对应。

GB 码，全称是 GB2312-80《信息交换用汉字编码字符集 基本集》，于 1980 年发布，是中文信息处理的国家标准，在中国内地及海外使用简体中文的地区（如新加坡等）是强制使用的唯

一中文编码。P-Windows 3.2 和苹果 OS 就是以 GB2312 为基本汉字编码，Windows 95/98 则以 GBK 为基本汉字编码，但兼容支持 GB2312。GB 码共收录 6763 个简体汉字、682 个符号，其中汉字部分：一级字 3755 个，以拼音排序，二级字 3008 个，以偏旁排序。该标准的制定和应用为规范、推动中文信息化进程起了很大作用。

GBK 编码是中国内地制定的、等同于 UCS 的新的中文编码扩展国家标准。GBK 工作小组于 1995 年 10 月开始，同年 12 月完成 GBK 规范。该编码标准兼容 GB2312，共收录汉字 21003 个、符号 883 个，并提供 1894 个造字码位，简、繁体字融于一库。

—— 来源于互联网

17.2 使用 convmv

convmv 的功能是转换文件名的编码，包括目录的转换。由于 Linux 系统中没有提供该工具，因此我们需要下载并且安装才可使用。

1. 下载安装

安装 convmv 只需要两步，make 和 make install：

```
[root@mfsmaster ~]http://www.j3e.de/linux/convmv/convmv-1.14.tar.gz
[root@mfsmaster convmv-1.14]# make
pod2man --section 1 --center=" " convmv | gzip > convmv.1.gz
[root@mfsmaster convmv-1.14]# make install
pod2man --section 1 --center=" " convmv | gzip > convmv.1.gz
mkdir -p /usr/local/share/man/man1/
mkdir -p /usr/local/bin/
cp convmv.1.gz /usr/local/share/man/man1/
install -m 755 convmv /usr/local/bin/
```

从最后一步可以看出，它只提供了一个命令，那就是 convmv。

2. 转换文件名编码

命令 convmv 的使用比较简单，下面是它的基本用法：

```
[root@mfsmaster ~]# convmv --help
Your Perl version has fleas #22111
convmv 1.14 - converts filenames from one encoding to another
Copyright (C) 2003-2008 Bjoern JACKE <bjoern@j3e.de>

This program comes with ABSOLUTELY NO WARRANTY; it may be copied or modified
under the terms of the GNU General Public License version 2 or 3 as published
by the Free Software Foundation.

 USAGE: convmv [options] FILE(S)
-f enc encoding *from* which should be converted
-t enc encoding *to* which should be converted
```

```
    -r recursively go through directories
    -i interactive mode (ask for each action)
    --nfc  target files will be normalization form C for UTF-8 (Linux etc.)
    --nfd  target files will be normalization form D for UTF-8 (OS X etc.)
    --qfrombe quiet about the "from" of a rename (if it screws up your terminal
e.g.)
    --qto  be quiet about the "to" of a rename (if it screws up your terminal
e.g.)
    --exec c   execute command instead of rename (use #1 and #2 and see man page)
    --list list all available encodings
    --lowmem   keep memory footprint low (see man page)
    --nosmart  ignore if files already seem to be UTF-8 and convert if posible
    --notest   actually do rename the files
    --replace  will replace files if they are equal
    --unescape convert%20ugly%20escape%20sequences
    --upperturn to upper case
    --lowerturn to lower case
    --parsable write a parsable todo list (see man page)
    --help print this help
```

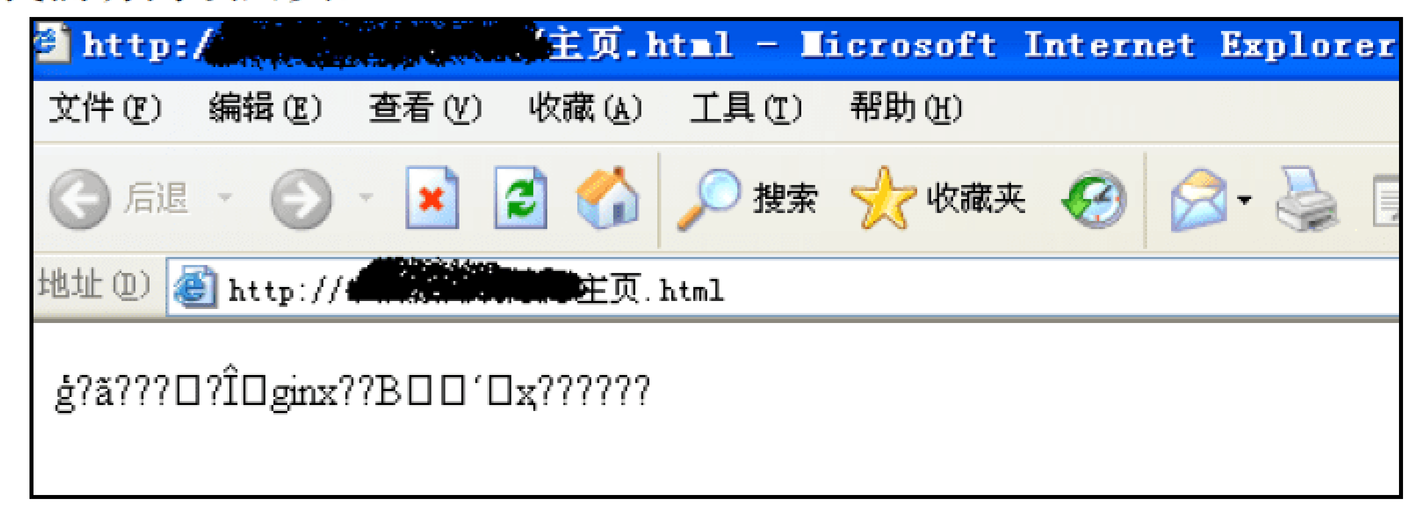
使用 convmv 命令将现有的文件从 GB2312 转换为 UTF-8：

```
[root@mfsmaster html]# convmv -f GB2312 -t UTF-8  --nosmart --notest ./*.*
mv "./??.html"  "./主页.html"
Ready!
[root@mfsmaster html]# ls
主页.html
```

顺便说一句，它也可以转换目录的编码，例如：

```
[root@mfsmaster html]# ls
????
[root@mfsmaster mm]# convmv -f GB2312 -t UTF-8  --nosmart --notest ./*
mv "./ü?"  "./图片"
Ready!
[root@mfsmaster mm]# ls
图片
```

好了，现在我们访问该网页：



可见，不再是 404 了，至少文件找到了，可网页还是乱码，我们来看一下在 Linux 系统中查看是什么情况：

```
[root@mfsmaster html]# more 主页.html

?o?????·??ginx£?B?????£?£?£?
```

同样是乱码，再看一下 Nginx 服务器的配置：

```
location / {
  root  /var/xx.com/html;
  index  index.html index.htm;
charset  utf-8;

}
```

这时，我们有两种选择，第一是将该网页内容也转换为 UTF-8；第二是重新设置 Nginx 的 charset，就是将 charset 设置为 GB2312。

我们先采取第一种方法将网页内容转换为 UTF-8，这时需要一个工具 enca，也许你会说 Linux 系统也带了一个 iconv 命令，也可以将文件内容做编码转换，但是 enca 提供了两个命令，一个是 enca，另一个是 enconv，通过 enca 可以先查看文件内容的编码，这是我们需要的，因此需要认识这个新的命令。

17.3 使用 enca

如果想查看一下文件名的编码，则需要 enca 命令，如果你的系统没有，需要先进行安装。

1. 下载并安装

enca 的安装很简单，就是简单的三步走：

```
[root@mfsmaster ~]# wget http://dl.cihar.com/enca/enca-1.13.tar.gz
[root@mfsmaster ~]#tar -zxvf enca-1.13.tar.gz
[root@mfsmaster ~]#cd enca-1.13.
[root@mfsmaster enca-1.13]#make
[root@mfsmaster enca-1.13]#make install
```

看一下它的用法：

```
[root@mfsmaster enca-1.13]# enca
Usage:  enca [-L LANGUAGE] [OPTION]... [FILE]...
enconv [-L LANGUAGE] [OPTION]... [FILE]...
```

有两个命令，前者用于查看文件内容的编码，而后者用于改变文件的编码。

2. 查看文件内容的编码

```
[root@mfsmaster html]# echo $LANG
zh CN.utf-8
[root@mfsmaster html]# ls
???.html
```

```
[root@mfsmaster html]# enca 'ls'
Unrecognized encoding
```

根据使用 enca 的经验，enca 有时候会将 GBK 编码识别失败，因此，该文将是 GBK 编码的嫌疑，而且需要注意的一点是，Windows 下的默认编码就是 GB2312。

3. 转换文件内容编码

我们使用该软件提供的 iconv 命令来进行转换：

```
[root@mfsmaster html]#  iconv  --from-code=GB2312 --to-code=UTF-8 主页.html  > 主页1.html
[root@mfsmaster html]# ls
主页1.html  主页.html
[root@mfsmaster html]# more  'ls'
::::::::::::::
主页1.html
::::::::::::::

你好，欢迎访问 Nginx，我是中文网页！！！
::::::::::::::
主页.html
::::::::::::::

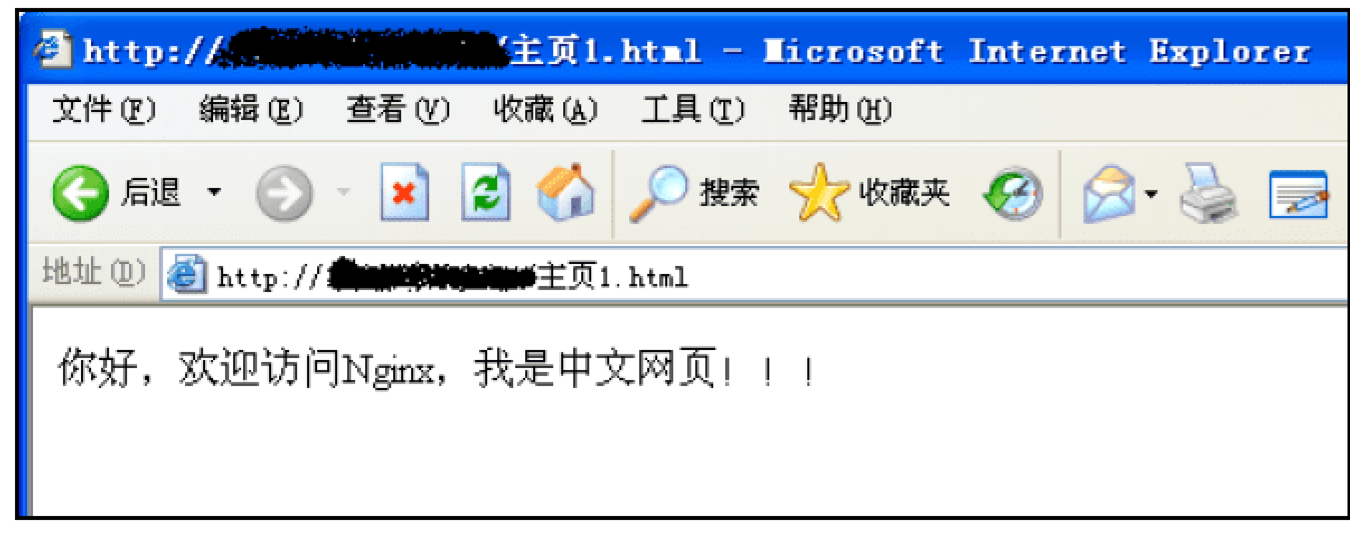
?o?????·??ginx£?B?????£?£?£?
```

再看一下文件内容的编码方式：

```
[root@mfsmaster html]# enca  'ls'
主页1.html: Universal transformation format 8 bits; UTF-8
  CRLF line terminators
主页.html: Simplified Chinese National Standard; GB2312
  CRLF line terminators
```

很明确"主页 1.html"的编码为 UTF-8，而且这次"主页.html"的编码也识别出了它的编码就是 GB2312。

看，我们的文件"主页 1.html"不再乱码了，现在访问"主页 1.html"：



可以了，不再是乱码了。

17.4 字符集设置模块

在前面我们看到了字符集设置指令 charset。下面我们具体了解 HttpCharsetModule 模块。

HttpCharsetModule 模块将会在响应头“Content-Type”字段中添加字符编码，而且，该模块还能够对数据进行重新编码，并将数据从一个编码转换为另一个编码。但是需要注意的是，重新编码这一操作仅是单方向的——从服务器端到客户端的数据可以重新编码，并且只能使用一种字节的编码方式来进行编码，就是说只能是一种编码方式。

1. 配置示例

```
charset windows-1251;
source_charset  koi8-r;
```

2. 指令

指令名称：charset

语法：charset encoding|off

默认值：charset off

使用环境：http，server，location，if in location

功能：该指令将会在响应头“Content-Type”字段中添加字符编码，以便指示编码。如果该指令指定的编码与指令 source_charset 指定的编码方式不同，那么将会重新编码。如果设定为 off，那么则不在响应头中添加“Content-Type”信息。

指令名称：charset_map

语法：charset_map encoding1 encoding2 {...}

默认值：no

使用环境：http，server，location

功能：该指令指定了一个从一种编码到另一种编码转换的转换表，同时会使用同样的数据创建一个反向转换表。编码的符号使用十六进制格式，如果在 80～FF 这个范围内没有相应的代码，它们将被标记为“？”。

例如：

```
charset_map koi8-r  windows-1251 {
                    C0  FE ; # small yu
                    C1  E0 ; # small a
                    C2  E1 ; # small b
                    C3  F6 ; # small ts
                    # ...
```

这是一个从 koi8-r 到 Windows-1251 转换的完整列表，该表位于 Nginx 的 conf/中的 koi-win 文件。

指令名称：override_charset

语法：override_charset on|off

默认值：override_charset off

使用环境：http，server，location，if in location

功能：该指令用于决定如果从代理服务器或者是 FastCGI 服务器传递来的响应头中已经有一个“Content-Type”头，那么 Nginx 是否对它进行重新编码。如果重新编码允许，那么将会在响应中使用 source_charset 设定的编码重新编码。然而需要注意的是，如果是从子查询中获取的响应，那么就不再依赖于指令 override_charset 的设置了，而总是将响应中的编码转换为基本的编码。

指令名称：source_charset

语法：source_charset encoding

默认值：no

使用环境：http，server，location，if in location

功能：该指令指定了响应的初始编码，如果这个编码与指令 charset 设置的不同，则会进行重新编码。

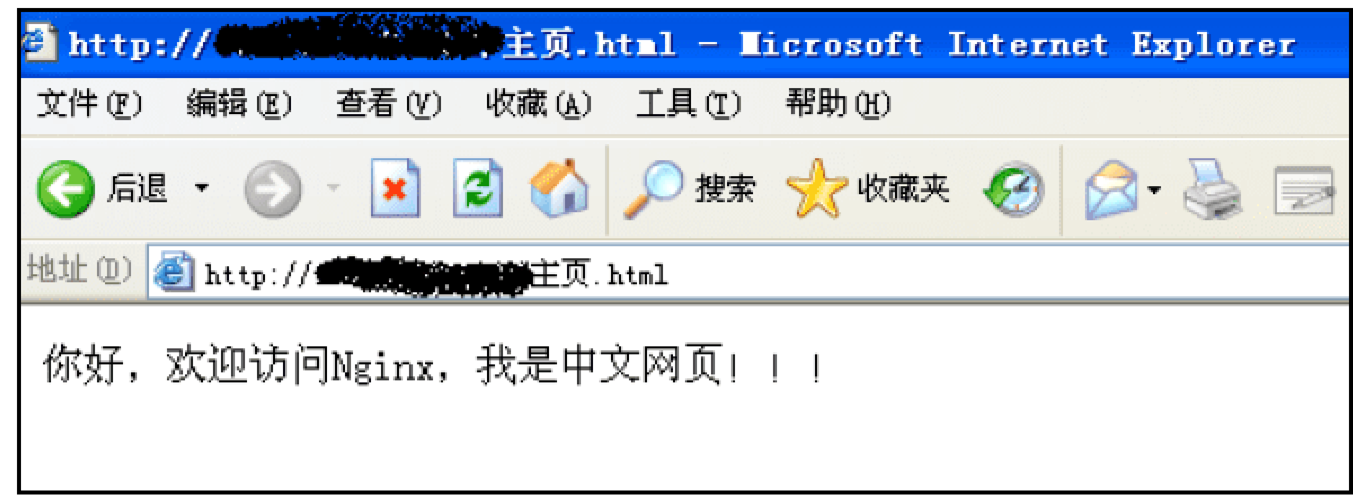
3. 使用实例

我们解决前面提过的乱码问题。在前面我们用转换文件内容的编码方式让文件内容不再乱码，便可以正常查看页面了。下面我们通过修改 Nginx 的配置文件内容，将 Nginx 的 charset 指令设置为 GB2312，例如：

```
    location / {
      root  /var/xx.com/html;
      index  index.html index.htm;
  charset   gb2312;

    }
```

重新启动 Nginx 服务器，再次访问（注意要清除浏览器缓存）“主页.html”：



好了，没问题了。

因此，是使用第一种方式进行文件内容编码转换还是通过修改 Nginx 的配置文件来实现，这取决于具体的情况。如果只是某几个文件，那么可以通过转换编码来实现，而如果某个网站或者是某个网站的某一部分都是用了某一种文件编码，那么就在 Nginx 服务器中相应的 server 或 location 中进行字符集设置。

第 18 章　网页压缩传输

在 Nginx 中与网页压缩相关的模块有两个：一个是 HttpGzipModule，另一个是 HttpGzipStaticModule。前者用于启用在文件传输过程中使用 gzip 压缩，而后者的作用是将一个文件以压缩的方式传递到一个支持压缩功能的客户端之前，首先检查是否已经存在相应的以“.gz”结尾的文件名格式，这样可以避免重复压缩而造成资源浪费。

因此，对于 Nginx 的网页压缩传输在 Nginx 中的配置是将其分为两种模式：一种是动态的，实时压缩输出（边压缩，边输出），而另一种则是静态的，找到同名文件的.gz 格式文件就输出。

18.1 HttpGzipModule

HttpGzipModule 用于在文件传输过程中启用 gzip 压缩，压缩率通过变量$gzip_ratio 来设定。

1. 配置示例

```
gzip on;
gzip_min_length  1000;
gzip proxied expired no-cache no-store private auth;
gzip types   text/plain application/xml;
gzip_disable "MSIE [1-6]\.";
```

2. 指令

HttpGzipModule 提供了以下两条指令。

指令名称：gzip

语法：gzip on|off

默认值：gzip off

使用环境：http，server，location，location 中的 if 区段

功能：启用或者禁用 gzip 压缩功能。

指令名称：gzip_buffers

语法：gzip_buffers number size

默认值：gzip_buffers 4 4k/8k

使用环境：http，server，location

功能：该指令用于指定存放被压缩响应的缓冲的数量和大小。如果没有设置大小，那么一个缓冲的大小等于一个页码的大小，具体页码的大小依赖于所在的系统平台，不是 4KB 就是 8KB。获取系统内存页码大小的方法很简单：

```
[root@s8 ~]# getconf PAGE_SIZE
4096
```

指令名称：gzip_comp_level

语法：gzip_comp_level 1..9

默认值：gzip_comp_level 1

使用环境：http，server，location

功能：该指令用于设定压缩级别，可以设定的级别从 1～9，1 是最小的压缩率，也是最快的，占用的 CPU 资源最少；而 9 是压缩率最多的，也是最慢的，占用的 CPU 资源最大。

指令名称：gzip_disable

语法：gzip_disable regex

使用环境：http，server，location

功能： 可以通过该指令对一些特定的用户代理不使用压缩功能，就可以使用正则表达式，但这需要 PCRE 的支持。该指令从 0.6.23.以后才开始提供。从 Nginx 0.7.63 版本后，可以使用“msie6”来禁止对 IE 5.5 和 IE 6 的压缩，而“SV1”（Service Pack 2）将会被忽略。

例如：

```
gzip_disable"msie6";
```

指令名称：gzip_http_version

语法：gzip_http_version 1.0|1.1

默认值：gzip_http_version 1.1

使用环境：http，server，location

功能：该指令用于决定对指定的 HTTP 请求协议版本进行压缩或者不压缩，其依赖于客户端的 HTTP 请求的版本。当使用 HTTP 1.0 协议时，Vary: Accept-Encoding 头没有设置，这样会导致代理缓存腐化（corruption），因此可以考虑使用 add_header 指令来添加它。同时也要注意到，无论使用 gzip 的哪个版本，Content-Length 头都没有设置。使用 1.0 版本时，Keepalive 将会无效，而 1.1 版本将会由 chunked 传递处理。

> 注意，该指令的默认值为 1.1，但是在某些抓取访问（例如 CDN）中可能会有问题，因此根据需要可以将其改为 1.0。

指令名称：gzip_min_length

语法：gzip_min_length length

默认值：gzip_min_length 0

使用环境：http，server，location

功能：该指令用于设置响应体的最小长度，单位为字节。如果响应体的长度低于指定的值，那么就不使用压缩。长度的决定从“Content-Length”头获取。

指令名称：gzip_proxied

语法：gzip_proxied [off|expired|no-cache|no-store|private|no_last_modified|no_etag|auth|any] ...

默认值：gzip_proxied off

使用环境：http，server，location

功能：该指令用于设置启用或禁用从代理服务器上收到的响应体 Gzip 压缩功能。该指令接受下列参数，有些可以组合使用。

- off/any：对所有的请求启用/禁用压缩功能。
- expired：如果 Expires header 阻止缓存，那么启用压缩。
- no-cache/no-store/private：如果 Cache-Control header 被设置为 no-cache、no-store 或 private，则启用压缩。
- no_last_modified：假如 Last-Modified header 没有设置，则启用压缩。
- no_etag：假如 ETag header 没有设置，则启用压缩。
- auth：假如设置了 Authorization header，则启用压缩。

指令名称：gzip_types

语法：gzip_types mime-type [mime-type ...]

默认值：gzip_types text/html

使用环境：http，server，location

功能：该指令用于设定除了默认的 text/html MIME 类型外，对其他的那些 MIME 类型也启用压缩功能。

指令名称：gzip_vary

语法：gzip_vary on|off

默认值：gzip_vary off

使用环境：http，server，location

功能：该指令用于设定是否向响应数据包添加 Vary: Accept-Encoding HTTP 头（header）。

> 需要注意的是，由于 bug 的原因，如果设置添加该头，那么会导致 IE 4～6 不缓存内容。

指令名称：gzip_window

语法：gzip_window size

默认值：MAX_WBITS，来源于 Zlib 库

使用环境：http，server，location

功能：该指令用于设置 Gzip 操作的窗口（Window）缓冲的大小（WindowBits 参数）。该指令所使用的值是由 Zlib 库调用的功能。

指令名称：gzip_hash

语法：gzip_hash size

默认值：MAX_MEM_LEVEL （前提是），来源于 Zlib 库

使用环境：http，server，location

功能：该指令用于设置分配给内部压缩状态（memLevel 参数）的内存总数。该指令所使用的值是由 Zlib 库调用的功能。

指令名称：postpone_gzipping

语法：postpone_gzipping size

默认值：0

使用环境：http，server，location

功能：在开始进行 Gzip 压缩前定义一个最小的数据门槛（threshold）。

指令名称：gzip_no_buffer

语法：gzip_no_buffer on|off

默认值：off

使用环境：http，server，location

功能：默认情况下，在将数据发送到客户端之前 Nginx 会等待，直到至少一个缓存（由 gzip_buffers 定义）被数据填满。如果开启该指令，那么会禁用缓存。

3. 使用实例

添加配置

在原有的配置区段中添加以下配置：

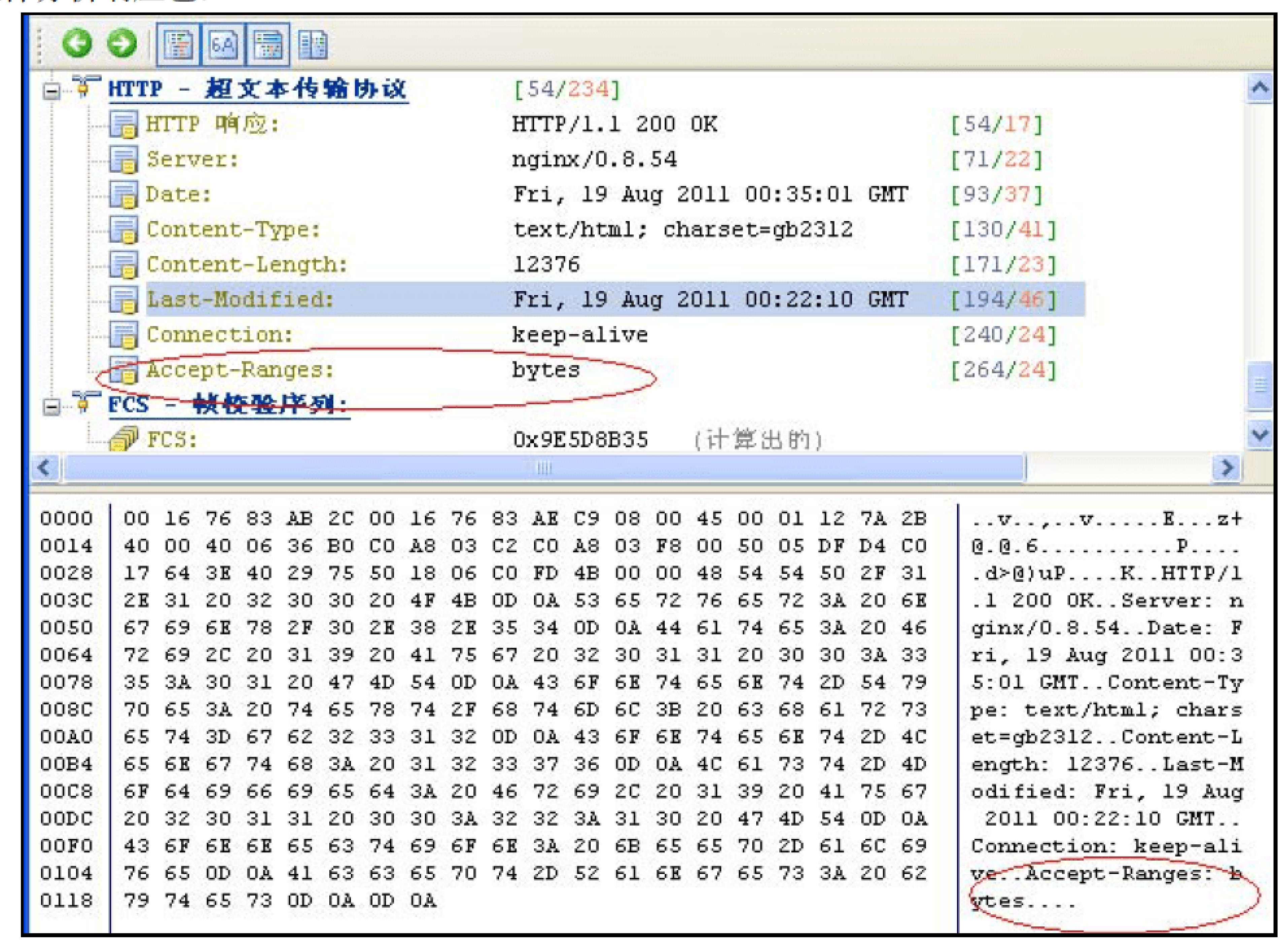
```
gzip on;
gzip_types   text/plain application/xml;
```

访问测试

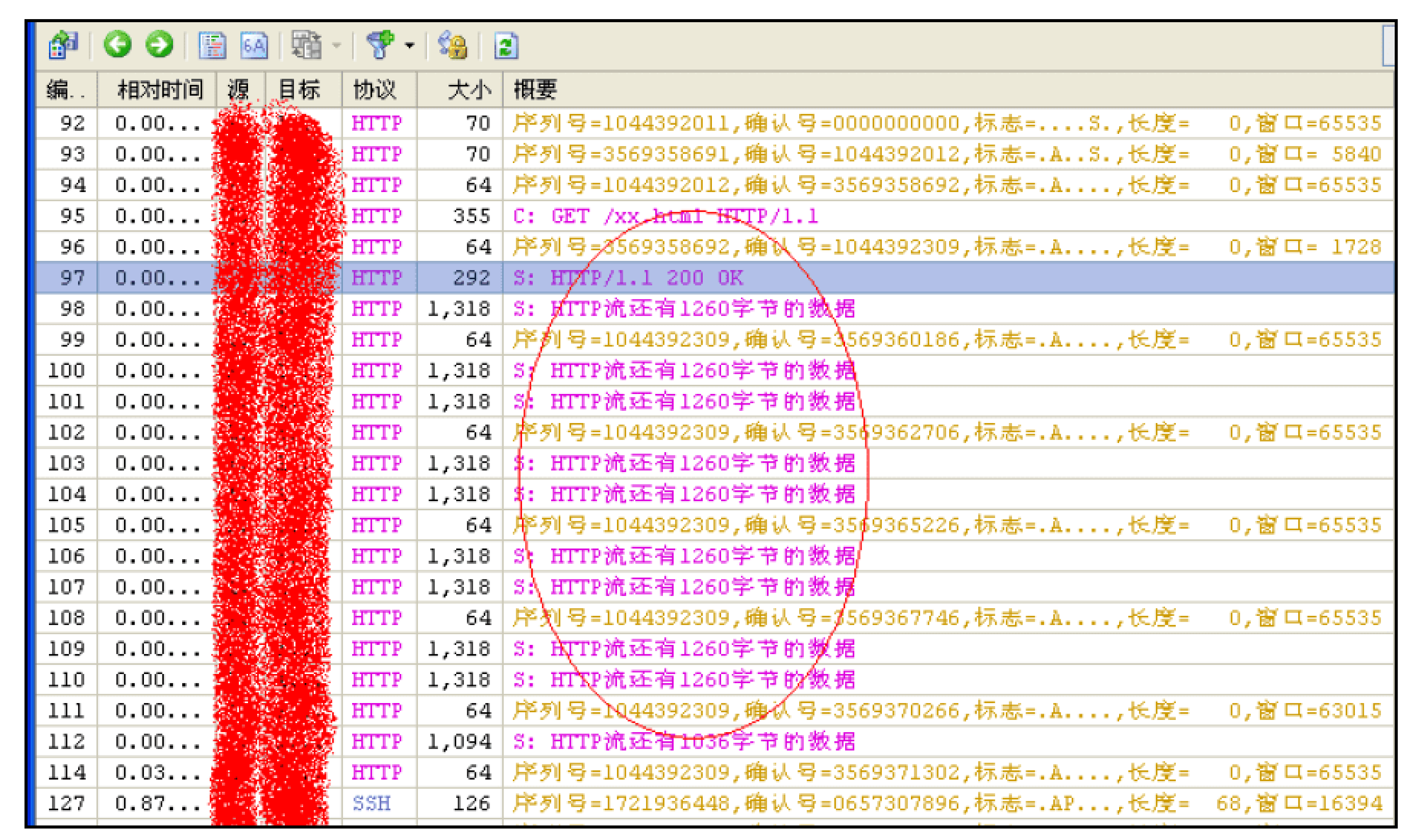
看一下我们要访问的网页（注意它的大小）：

```
[root@mfsmaster html]# ll
-rw-r--r--  1 root root 12376  8 月 19 08:22 index.html
```

在添加以上配置后，先不要重新启动或者重新载入配置文件，然后访问该网页。用协议分析软件分析响应包：

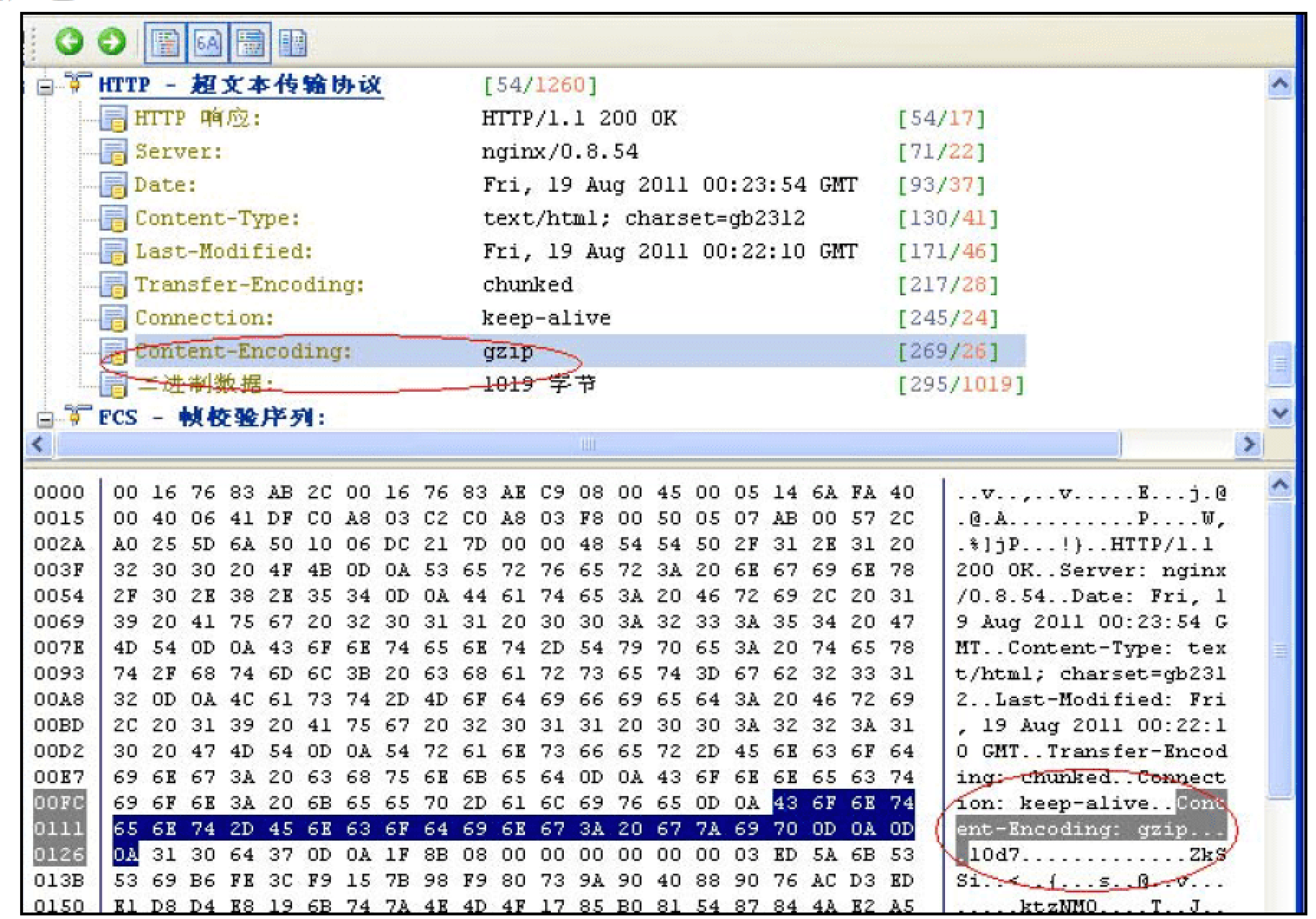


注意，在这里没有使用“Accept-encoding: gzip”，而是返回“Accept-Ranges: bytes”表示支持断点续传，同时注意到它的 Content-Length 为 12376，看下面的截图：

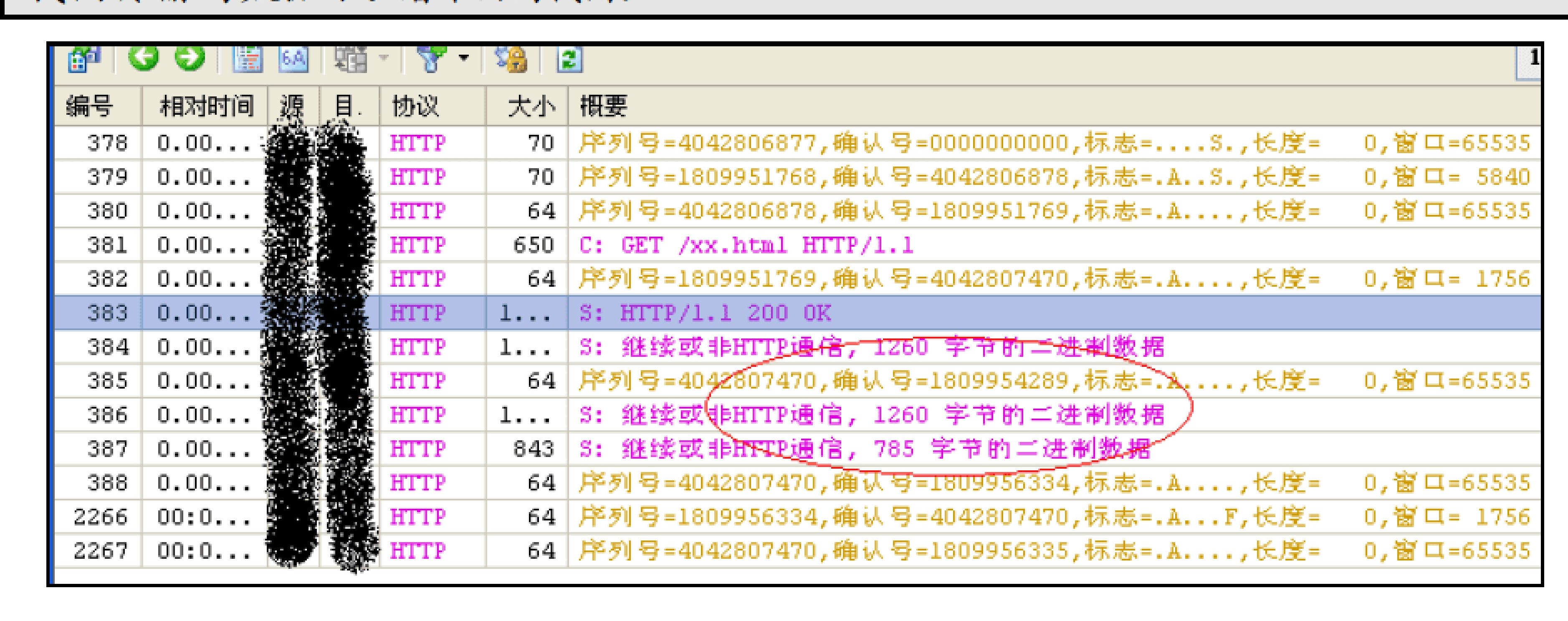


在这个截图中，我们计算一下传输的字节：1260*9+1036=12376。

然后重新载入 Nginx 的配置，再次进行访问，注意要访问同一页面（记得清除缓存），并分析响应包：



> 注意图中圈起的部分，这是服务器的返回信息，说明我们这次使用的是压缩传输，即“Content-Encoding: gzip”；另外，要注意这里的“二进制数据 1019 字节”这已经是我们传输的数据了。看下面的截图：



编号	相对时间	源	目.	协议	大小	概要
378	0.00...	[redacted]	[redacted]	HTTP	70	序列号=4042806877,确认号=0000000000,标志=....S.,长度= 0,窗口=65535
379	0.00...	[redacted]	[redacted]	HTTP	70	序列号=1809951768,确认号=4042806878,标志=.A..S.,长度= 0,窗口= 5840
380	0.00...	[redacted]	[redacted]	HTTP	64	序列号=4042806878,确认号=1809951769,标志=.A....,长度= 0,窗口=65535
381	0.00...	[redacted]	[redacted]	HTTP	650	C: GET /xx.html HTTP/1.1
382	0.00...	[redacted]	[redacted]	HTTP	64	序列号=1809951769,确认号=4042807470,标志=.A....,长度= 0,窗口= 1756
383	0.00...	[redacted]	[redacted]	HTTP	1...	S: HTTP/1.1 200 OK
384	0.00...	[redacted]	[redacted]	HTTP	1...	S: 继续或非HTTP通信，1260 字节的二进制数据
385	0.00...	[redacted]	[redacted]	HTTP	64	序列号=4042807470,确认号=1809954289,标志=.A....,长度= 0,窗口=65535
386	0.00...	[redacted]	[redacted]	HTTP	1...	S: 继续或非HTTP通信，1260 字节的二进制数据
387	0.00...	[redacted]	[redacted]	HTTP	843	S: 继续或非HTTP通信，785 字节的二进制数据
388	0.00...	[redacted]	[redacted]	HTTP	64	序列号=4042807470,确认号=1809956334,标志=.A....,长度= 0,窗口=65535
2266	00:0...	[redacted]	[redacted]	HTTP	64	序列号=1809956334,确认号=4042807470,标志=.A...F,长度= 0,窗口= 1756
2267	00:0...	[redacted]	[redacted]	HTTP	64	序列号=4042807470,确认号=1809956335,标志=.A....,长度= 0,窗口=65535

> 在这个截图中传输的字节数为：1260*2+785+1019=4324。比“.gz”格式多 292 个字节。后面会讲到“.gz”格式的传输。

18.2 HttpGzipStaticModule

在从磁盘向支持 gzip 的客户端提供一个文件时，HttpGzipStaticModule 将会在同样的目录（或者叫位置）中查找与请求文件名相同的、以“.gz”格式结尾的文件，这个文件被称为文件的“预压缩格式”。之所以称为“预压缩格式”，是因为 Nginx 不会对该文件进行压缩，即使该文件被访问之后也不会产生“.gz”格式的文件，因此需要我们自己压缩。那么这种机制的作用是什么呢？很简单，这么做的原因是为了避免每次请求都将对同一个文件进行压缩。

> ngx_http_gzip_static_module 从 Nginx 0.6.24 版本开始提供，但是在默认安装中它是不会被编译安装的，因此，在编译时需要指定--with-http_gzip_static_module 选项。

1. 配置示例

```
gzip static on;
gzip http version  1.1;
gzip proxiedexpired no-cache no-store private auth;
gzip disable"MSIE [1-6]\.";
gzip_vary  on;
```

2. 指令

指令名称：gzip_static

语法：gzip_static on|off

默认值：gzip_static off

使用环境：http，server，location

功能：用于启用 HttpGzipStaticModule。需要注意的是，确定压缩版本和非压缩版本的时间戳要匹配，以便提供最新的内容。

以下指令参考 NginxHttpGzipModule 模块。

- 指令名称：gzip_http_version
- 指令名称：gzip_proxied
- 指令名称：gzip_disable
- 指令名称：gzip_vary

3. 使用实例

在下面的例子中我们先为现有的网页 index.html 生成一个“.gz”格式的文件，即 index.html.gz，然后测试访问；再对 index.html 文件进行修改，并访问测试。

添加配置

```
  gzip on;
  gzip_types   text/plain application/xml;
  gzip_static on;
```

访问测试

生成 index.html 文件的另一个格式 index.html.gz：

```
[root@mfsmaster html]# ls
index.html
[root@mfsmaster html]#cat index.html
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx! 哈哈！！！</h1></center>
</body>
</html>
[root@mfsmaster html]# gzip -c index.html  > index.html.gz
[root@mfsmaster html]# ls
index.html  index.html.gz
```

确定文件的访问时间：

```
[root@mfsmaster html]# stat index.*
  File: 'index.html'
  Size: 167 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394667 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 16:45:20.339995192 +0800
Modify: 2011-08-18 16:44:16.746662848 +0800
Change: 2011-08-18 16:44:16.746662848 +0800
  File: 'index.html.gz'
```

```
  Size: 151 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394635 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 16:45:20.338995344 +0800
Modify: 2011-08-18 16:45:20.339995192 +0800
Change: 2011-08-18 16:45:20.339995192 +0800
```

访问该文件：



查看文件的访问时间：

```
[root@mfsmaster html]# stat index.*
  File: 'index.html'
  Size: 167 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394667 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 16:45:20.339995192 +0800
Modify: 2011-08-18 16:44:16.746662848 +0800
Change: 2011-08-18 16:44:16.746662848 +0800
  File: 'index.html.gz'
  Size: 151 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394635 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 16:59:01.040229792 +0800
Modify: 2011-08-18 16:45:20.339995192 +0800
Change: 2011-08-18 16:45:20.339995192 +0800
```

我们比较以下两个文件的访问时间戳。确切地说，我们的访问是由“index.html.gz”文件提供的。

下面将对 index.html 文件进行修改：

```
[root@mfsmaster html]# vi index.html

<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx! 哈哈！！！哈哈！！！</h1></center>
</body>
</html>
```

查看文件的访问时间：

```
[root@mfsmaster html]# stat index.*
  File: 'index.html'
  Size: 183 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394671 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 18:02:40.022656216 +0800
Modify: 2011-08-18 18:02:40.022656216 +0800
Change: 2011-08-18 18:02:40.023656064 +0800
  File: 'index.html.gz'
  Size: 151 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394635 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 16:59:01.040229792 +0800
Modify: 2011-08-18 16:45:20.339995192 +0800
Change: 2011-08-18 16:45:20.339995192 +0800
```

再次访问该网页。得到的页面和原来的一样，再查看文件的访问时间戳（如果你也是在做测试，那么你需要将 IE 浏览器的缓存清除）：

```
[root@mfsmaster html]# stat index.*
  File: 'index.html'
  Size: 183 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394671 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 18:02:40.022656216 +0800
Modify: 2011-08-18 18:02:40.022656216 +0800
Change: 2011-08-18 18:02:40.023656064 +0800
  File: 'index.html.gz'
  Size: 151 Blocks: 8  IO Block: 4096   一般文件
Device: fd00h/64768dInode: 5394635 Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 18:09:11.569132104 +0800
Modify: 2011-08-18 16:45:20.339995192 +0800
Change: 2011-08-18 16:45:20.339995192 +0800
```

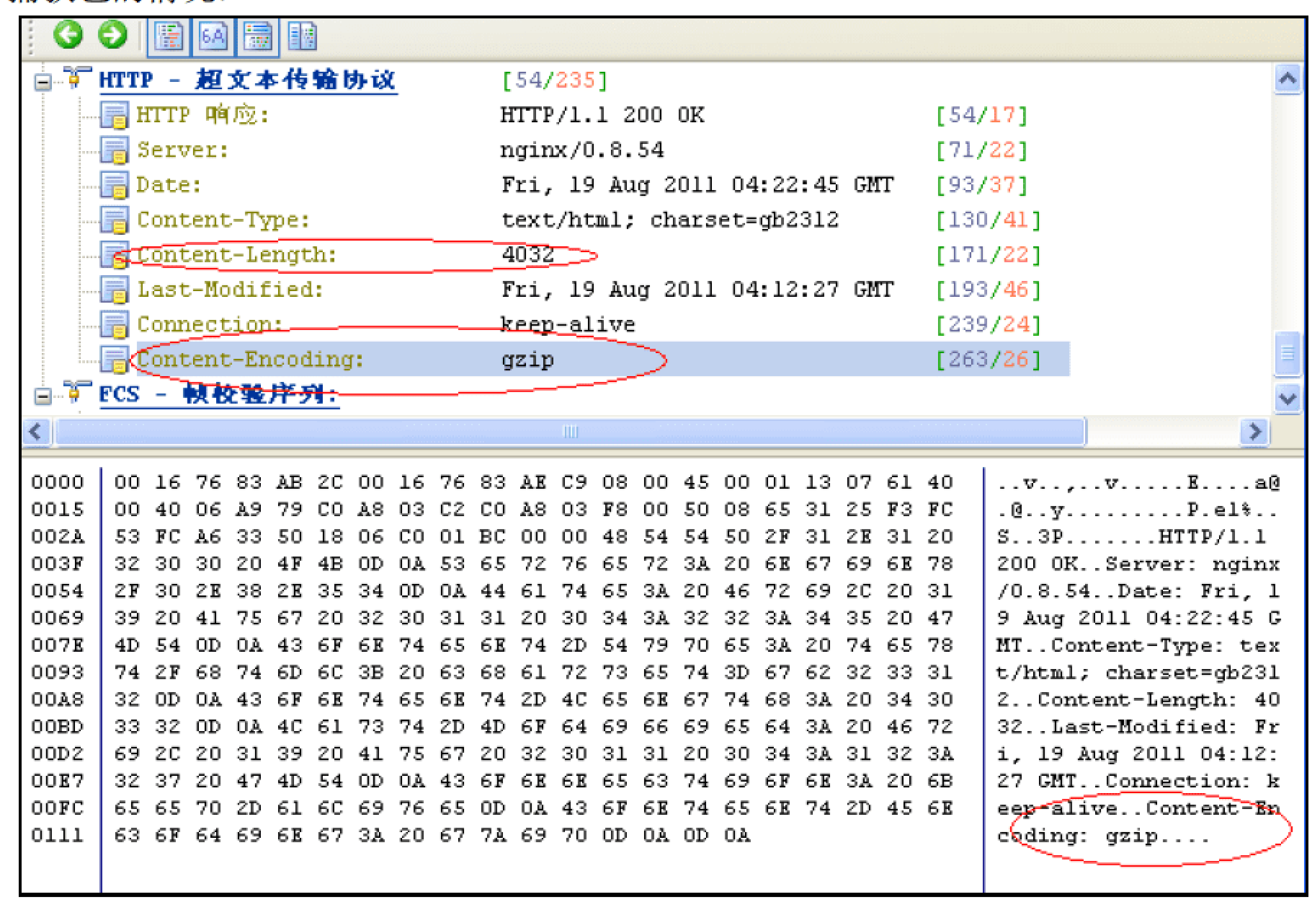
相信你一定看清楚了，是由文件“index.html.gz”来提供访问的，Nginx 并没有提供最新时间的“index.html”文件。你要还不信，那就将文件“index.html.gz”删除再访问，网页绝对是最新版本。在此就不再举例了。

我们来看，以下访问情况：

```
[root@mfsmaster html]# ll
总用量 52
-rw-r--r--  1 root root   152  8月 18 19:07 index.html
-rw-r--r--  1 root root   151  8月 18 16:45 index.html.gz.old
-rw-r--r--  1 root root 12376  8月 19 08:22 xx.html
```

```
-rw-r--r--  1 root root  4032  8 月 19 12:12 xx.html.gz
```

在这里为了说明访问情况，我们访问 http://www.xx.com/xx.html，页面就不再截取了。下面看捕获包的情况：



看一下图中被圈起的部分。下面是传输的数据包：

数据包

编号	相对...	源	目标	协议	大小	概要
91	0.0...	1.	1...	HTTP	70	序列号=1409066154,确认号=0000000000,标志=....S.,长度= 0,窗口=65535
92	0.0...	1.	1...	HTTP	70	序列号=0824570875,确认号=1409066155,标志=.A..S.,长度= 0,窗口= 5840
93	0.0...	1.	1...	HTTP	64	序列号=1409066155,确认号=0824570876,标志=.A....,长度= 0,窗口=65535
94	0.0...	1.	1...	HTTP	450	C: GET /xx.html HTTP/1.1
95	0.0...	1.	1...	HTTP	64	序列号=0824570876,确认号=1409066547,标志=.A....,长度= 0,窗口= 1728
96	0.0...	1.	1...	HTTP	293	S: HTTP/1.1 200 OK
97	0.0...	1.	1...	HTTP	1,318	S: 继续或非HTTP通信, 1260 字节的二进制数据
98	0.0...	1.	1...	HTTP	64	序列号=1409066547,确认号=0824572371,标志=.A....,长度= 0,窗口=65535
99	0.0...	1.	1...	HTTP	1,318	S: 继续或非HTTP通信, 1260 字节的二进制数据
100	0.0...	1.	1...	HTTP	1,318	S: 继续或非HTTP通信, 1260 字节的二进制数据
101	0.0...	1.	1...	HTTP	64	序列号=1409066547,确认号=0824574891,标志=.A....,长度= 0,窗口=65535
102	0.0...	1.	1...	HTTP	310	S: 继续或非HTTP通信, 252 字节的二进制数据
104	0.1...	1.	1...	HTTP	64	序列号=1409066547,确认号=0824575143,标志=.A....,长度= 0,窗口=65283
2219	00:...	1.	1...	HTTP	64	序列号=0824575143,确认号=1409066547,标志=.A...F,长度= 0,窗口= 1728
2220	00:...	1.	1...	HTTP	64	序列号=1409066547,确认号=0824575144,标志=.A....,长度= 0,窗口=65283
2252	00:...	1.	1...	HTTP	64	序列号=1409066547,确认号=0824575144,标志=.A...F,长度= 0,窗口=65283
2253	00:...	1.	1...	HTTP	64	序列号=0824575144,确认号=1409066548,标志=.A....,长度= 0,窗口= 1728

字节数：1260*3+252=4032，绝对访问的是 xx.html.gz 页面！

说了这么多，其实我们要明白的是压缩传输的好处，绝对节省带宽。我们再计算一下。看下面的算式：

（12376-4032）/12376=67.42%

（12376-4324）/12376=65.06%

因此，得出的结论是：页面压缩传输后节省了大约 60%的带宽。

第 19 章　控制 Nginx 如何记录日志

在 Nginx 服务器中，如果想对日志输出进行控制还是很容易的。Nginx 服务器提供了一个 HttpLogModule 模块，可以通过它来设置日志的输出格式，然而在具体的生产环境中却很少使用，通常都将访问日志关闭。但是日志从来都被看做是信息的黄金库，因此必要的时候还是有必要使用的。

1. 配置示例

```
log format gzip '$remote addr - $remote user [$time local]  '
'"$request" $status $bytes sent '
'"$http_referer" "$http_user_agent" "$gzip_ratio"';
access_log /spool/logs/nginx-access.log  gzip  buffer=32k;
```

2. 指令

HttpLogModule 模块提供了三条指令，可以用它们来设置日志的格式、缓存等。

指令名称：access_log

语法：access_log path [format [buffer=size]] | off

默认值：access_log log/access.log combined

使用环境：http，server，location

功能：该指令用于为访问日志设置存放缓存的路径、格式，以及缓存大小。如果在当前的区段中将该指令的值设置为 off，那么将会清除从上级继承而来的所有 access_log 的参数。在使用该指令时，如果没有指定日志格式，则会使用默认的“combined”格式。在默认设置中，缓存是关闭的，这一点要注意。

对于 0.7.4 版本以上的 Nginx，在日志文件中，路径可以包含变量，但是这样的日志有一定的限制：

- 对于该目录路径 worker 用户必须有建立新文件的权限；
- 缓存区不会工作，就是说不能够使用缓存。

对于每一条日志条目，日志文件被打开，写入记录后就迅速地关闭。但是，常用文件的描述符却被存储在由指令 open_log_file_cache 指定的缓存中。在这种情况下，如果产生日志轮换，那么必须注意的是，由指令 open_log_file_cache 设置的缓存，存储在缓存中的有效条目仍然会将日志写往该缓存条目生成时的变量（就是该变量指定的目录、文件）中去。

Nginx 支持强大的日志访问记录，每一个 location 中都可以有自己的日志记录，在同一时刻访问记录还可以输出到多个日志中。在后面的实例部分将有介绍。

指令名称：log_format

语法：log_format name format [format ...]

默认值：log_format combined "..."

使用环境：http，server

功能：该指令描述了日志条目的格式。可以在格式中使用一般的变量，也可以使用仅在写入日志的那一刻的变量（也叫过程变量）。这些变量如下。

- $body_bytes_sent：该变量的值是一个字节数，也就是传递到客户端的字节数减去响应头后的大小。该变量是兼容 Apache mod_log_config 的%B 参数；
- $bytes_sent：传递到客户端的字节数；
- $connection：连接数；
- $msec：写入日志条目的当前时间（精确到百万分之一秒）；
- $pipe：如果是 HTTP pipe 技术，那么这里将会是一个“p”字符；
- $request_length：请求体的长度；
- $request_time：请求时间，这个时间是指 Nginx 在该请求上花费的时间，单位精确到毫秒（在低于 0.5.19 的版本中只使用到秒）；
- $status：响应的状态代码；
- $time_iso8601：使用 ISO 8601 格式的时间，例如 2011-03-21T18:52:25+03:00 （从 0.9.6 版本开始提供了该变量）；
- $time_local：在日志中写入服务器本地的时间。

还有一些其他变量，例如：

- $sent_http_content_range：这是一个传递到客户端的头，这类头的前缀为“sent_http_”。
- “upstream_http_”：这是由 upstream 模块产生的日志，因此它的前缀将会是该前缀。所以有些变量值还有可能是其他模块产生的变量。

在这个模块中，预定义了一个叫做“combined”的日志格式：

```
log_format combined '$remote_addr - $remote_user [$time_local]  '
'"$request" $status $body_bytes_sent '
'"$http_referer" "$http_user_agent"';
```

指令名称：open_log_file_cache

语法：open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time] | off

默认值：open_log_file_cache off

使用环境：http，server，location

功能：该指令用于设置缓存。该缓存用于存储带有变量的日志文件路径而又频繁使用的文件描述符，这些被频繁使用的文件描述符将会被存储在缓存中。

指令选项：

- max - ：该选项用来设置在缓存中可以存储的最大描述符数量。它通过最近最少使用（LRU）算法来移除缓存条目。
- inactive - ：该选项用来设置一个时间间隔。在这个时间间隔之后，没有被命中的文件描述符将会被移除，默认值是 10 秒。
- min_uses - ：该选项用来设置访问次数。在一定的时间间隔内（这个间隔通过 inactive 选项获取），一个文件描述符至少被访问多少次后就可以将该描述符放在缓存中，默认值为 1，即访问一次便缓存。
- valid - ：该选项用于设置检查同名文件存在的时间，默认值是 60 秒。

- off - ：关闭缓存。

例如：

```
open_log_file_cache max=1000 inactive=20s min_uses=2 valid=1m;
```

3. 使用实例

在这里我们列举了两个典型的例子。在第一个例子中，实现了自定义日志和多个日志记录；在第二个例子中，实现了日志文件路径、文件名称变量的使用。

实例 1

在 Nginx 中添加以下设置：

```
http {
  include    mime.types;
  default_type  application/octet-stream;
  sendfileon;
  keepalive_timeout  65;

  log_format custom $time_local | $server_name |
              $request_length | $bytes_sent;
  log_format apache-style '$remote_addr - $remote_user [$time_local]  '
   '"$request" $status $body_bytes_sent '
   '"$http_referer" "$http_user_agent"';
open_log_file_cache max=1000 inactive=20s min_uses=1 valid=1m;

  server {
listen   80;
server_name  localhost;

location /rmb {
  root   html;
  index  index.html index.htm;
  access_log logs/custom.log custom buffer=32k;
  access_log logs/apache-style.log  apache-style buffer=32k;
}

 location /rmj {
  root   html;
  index  index.html index.htm;
  access_log logs/custom.log custom buffer=32k;
  access_log /var/log/nginx/custom.log custom buffer=32k;
}
}
}
```

访问/rmb，并监控日志。以下是 apache-style 格式的日志：

```
[root@mail logs]# tail -f apache-style.log
192.168.3.248 - - [23/Aug/2011:09:20:33 +0800] "GET /rmb/ HTTP/1.1" 200
33 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
Gecko/20101026 Firefox/3.6.12 GTB7.1
```

custom 格式的日志：

```
[root@mail logs]# tail -f custom.log

23/Aug/2011:09:20:33 +0800|localhost|739|247
```

访问/rmb，并监控日志。以下均为 custom 格式的日志：

```
[root@mail logs]# tail -f custom.log

23/Aug/2011:09:22:10 +0800|localhost|739|248
```

```
[root@mail logs]# tail -f  /var/log/nginx/custom.log

23/Aug/2011:09:22:10 +0800|localhost|739|248
```

实例 2

在 Nginx 中添加以下设置：

```
http {
  include   mime.types;
  default_type  application/octet-stream;
  sendfileon;
  keepalive timeout  65;

  log_format custom $time_local | $server_name |
               $request_length | $bytes_sent;

  access_log logs/$server_name/custom.log custom;
open_log_file_cache max=1000 inactive=20s min_uses=1 valid=1m;

server {
  listen   80;
  server name  www.xx.com;
  location /rmb {
    root   html;
    index  index.html index.htm;
  }
}

server {
  listen   80;
  server_name  app1.xx.com;
  location /rmj {
```

```
    root   html;
    index  index.html index.htm;
  }
 }
   }
```

在这个配置文件中我们使用了变量。因为 Nginx 不会去创建目录，所以，想将将日志文件写入相应的域名目录中，必须将相应的域名目录创建好，并且具有写的权限，例如：

```
[root@mail logs]# ll

drwxrwxrwx 2 root root  4096 Aug 23 10:18 app1.xx.com
drwxrwxrwx 2 root root  4096 Aug 23 10:14 www.xx.com
```

如果某个域名目录没有创建，那么日志将不会写入，这对于某些需求也不错——根据域名有选择地记录日志。

对这两个域名访问后将会创建相应的日志，看一下访问后的目录结构：

```
[root@mail logs]# tree
|-- access.log
|-- app1.xx.com
|   '-- custom.log
|-- error.log
|-- nginx.pid
'-- www.xx.com
'-- custom.log

2 directories, 5 files
```

如果一定要记录日志，使用这种配置，可能会更方便：

```
access_log  logs/$server_name.custom.log custom;
```

4. 日志切割

Nginx 服务器中关于日志切割这一项并没有被解决，也不知道它们的开发人员是怎么想的，但是我想这是没必要的，在本节一开始就说过，在生产环境下的访问日志都是关闭的，即设置为“access_log off”。但是作为日志管理的一部分，则需要将其完善，在系统级完成该功能，也就是我们的 shell 脚本了。

这是 LNMP 套装中的一个日志切割脚本：

```
#!/bin/bash
#function:cut nginx log files for lnmp v0.5 and v0.6
#author: http://lnmp.org

#set the path to nginx log files
log_files_path="/home/wwwlogs/"
log_files_dir=${log_files_path}$(date -d "yesterday" +"%Y")/$(date -d
"yesterday" +"%m")
#set nginx log files you want to cut
```

```
log_files_name=(access vpser licess)
#set the path to nginx.
nginx_sbin="/usr/local/nginx/sbin/nginx"
#Set how long you want to save
save days=30

############################################
#Please do not modify the following script #
############################################
mkdir -p $log_files_dir

log files num=${#log files name[@]}

#cut nginx log files
for((i=0;i<$log_files_num;i++));do
mv ${log_files_path}${log_files_name[i]}.log
${log_files_dir}/${log_files_name[i]}_$(date -d "yesterday" +"%Y%m%d").log
done

#delete 30 days ago nginx log files
find $log files path -mtime +$save days -exec rm -rf {} \;

$nginx_sbin -s reload
```

根据实际需要进行修改使用就可以了，我没有使用过该脚本。

第 20 章　map 模块的使用

map 模块允许我们分类或者是将一组值映射到另一组不同的值，并将结果存储在变量中。map 指令用于创建变量，但是仅在变量被访问时才会执行映射操作。由于该模块在处理请求上并没有引用变量，因此，在性能上没有任何损失。

1. 配置示例

```
map  $http host  $name  {
  hostnames;

  default  0;

  example.com  1;
  *.example.com1;
  test.com 2;
  *.test.com   2;
  .site.com3;
wap.*4;

 }
```

对于 map 模块，一个典型的使用映射的例子是代替一个含有很多服务器的/location 或者重定向指令：

```
map $uri $new {
  default   http://www.domain.com/home/;

  /aa   http://aa.domain.com/;
  /bb   http://bb.domain.com/;
  ^/cc/ (?<suffix>.*) $  http://cc.domain.com/$suffix;
  /john http://my.domain.com/users/john/;
}

server {
  server_name   www.domain.com;
  rewrite  ^$new   redirect;
}
```

2. 指令

map 模块提供了三条指令。需要注意的是，这三条指令都是在 http 区段进行配置。

指令名称：map

语法：map $var1 $var2 {…}

默认值：none

使用环境：http

功能：该指令用于定义一个设置变量的表。该表有两列，即模式（pattern）和值。模式可以是一个简单的字符串或者是一个正则表达式，正则表达式的前导符为“~”。

例如：

```
map $uri $myvalue {
/aa   /mapped_aa;
~^/aa/(?<suffix>.*)$  /mapped_bb/$suffix;
}
```

如果在指定的模式开始处有一个“~”（波浪字符），但是它又不是正则表达式的一部分，那么可以在波浪字符之前添加一个反斜线（\），例如：

```
map $http_referer $myvalue {
Mozilla1234;
\~Mozilla  5678;
}
```

map 指令有三个特殊值：default、hostnames 和 include。

- default：指定无匹配内容时使用的默认值。
- include：一个包含有值的文件，可以多次使用 include。
- hostnames：允许使用通配符来匹配主机名。

例如：

```
*.example.com  1;
```

这种格式实际并没有代替以下两个条目：

```
example.com1;
*.example.com  1;
```

但是可以通过表达式表示这两个条目：

```
.example.com   1;
```

指令名称：map_hash_max_size

语法：map_hash_max_size   size

默认值：map_hash_max_size 2048

使用环境：http

功能：该指令用于设置哈希表的最大值，该哈希表与 map 映射表对应，为了能够使得 Nginx 更加快速地处理请求，Nginx 使用了哈希表。

指令名称：map_hash_bucket_size

语法：map_hash_bucket_size n

默认值：map_hash_bucket_size 32/64/128

使用环境：http

功能：该指令用于设置哈希表的最大值，默认值依赖于处理器缓存行（或页）。

3. 使用实例

在这里我们来看两个例子，一个是关于 URI 的替换，即虚拟“目录”替换，第二个是虚拟主机的例子，具体来说就是，系统目录替换。

实例 1

使用 map 的情况不是很多，但它确实比 rewrite 模块简单，同样也就没有 rewrite 的功能强大。下面的这个配置来自于本节开始的配置示例。

在这个配置文件中，我们将 mail、bbs 和个人网站的访问分别转到了相应的地址，但对于正则表达式的使用，这里做了调整，我们看一下配置：

```
map $uri $r {
  defaulthttp://www.xx.com/news/;

  /mail  http://mail.xx.com/;
  /bbs   http://bbs.xx.com/;
  ^/f/ (?<file>.*) $   http://f.xx.com/;
  /hjq   http://user.xx.com/users/hjq/;
}

server {
  server name   www.xx.com;
  rewrite  ^$r$1   redirect;
}
```

注意黑体字部分。

访问测试

访问 http://www.xx.com/mail，将会跳到 http://mail.xx.com，同样 bbs 的访问和个人网站的访问也是如此；我们要说的是“^/f/（?<file>.*）$”，例如 我们访问 http://www.xx.com/f/rk/p10258.html 页面时，将会进入 http://f.xx.com/rk/p10258.html。访问的页面就不再截取了，看一下访问日志。

这是 www.xx.com 的 Nginx 日志：

```
192.168.10.128- - [24/Aug/2011:15:00:00 +0800] "GET /f/rk/p10258.html HTTP/1.1" 302 160 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
```

我们在这个日志中可以看得到客户端发送来的请求是“**GET/f/rk/p10258.html HTTP/1.1**”，而状态码为 302，说明访问被重定向了。根据我们在 Nginx 的配置，该请求应该是被重定向到了 f.xx.com，我们看一下它的日志：

```
192.168.10.128 - - [24/Aug/2011:13:54:07 +0800] "GET /rk/p10258.html HTTP/1.1" 200 134 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
```

```
3.0.04506.30; CIBA) "
```

这也是一台 Nginx 服务器，它被请求的是 **"GET /rk/p10258.html HTTP/1.1"**，访问状态码为 200，说明我们的访问成功，即重定向成功。

实例 2

这是一个关于虚拟主机的例子，用 map 来进行目录映射，看一下配置：

```
map  $http host  $name  {
 www.xx.comxx.com/www;
 mail.xx.com   xx.com/mail;
 www.yy.comyy.com/www;
}

server {
server name  www.xx.com mail.xx.com www.yy.com;

location / {
root  /www/$name;
   }
}
```

这是目录结构：

```
[root@mail /]# tree /www
/www
|-- xx.com
|   |-- mail
|   |   '-- index.html
|   '-- www
|   '-- index.html
'-- yy.com
'-- www
'-- index.html

5 directories, 3 files
```

访问测试

根据客户端访问的具体域名，Nginx 将会根据 **$name** 找到相应的目录。例如，访问 http://mail.xx.com，那么在 Nginx 的日志中将会是以下内容：

```
192.168.10.128 - - [24/Aug/2011:15:44:05 +0800] "GET / HTTP/1.1" 200 23 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler
4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
```

无论访问/mail.xx.com，www.yy.com 还是 www.xx.com，在日志中写的都一样。因此，如果确实需要记录日志，那么可以在 server 中添加以下指令：

```
access_log /var/log/nginx/$host.access.log;
```

那么最后的结果将是：

```
[root@mail conf]# tree  /var/log/nginx/
/var/log/nginx/
|-- mail.xx.com.access.log
|-- www.xx.com.access.log
'-- www.yy.com.access.log

0 directories, 3 files
```

第 21 章　Nginx 预防应用层 DDoS 攻击

如果你的 Nginx 服务器规定只能同时连接一个用户，即并发连接数为 1，那么就不用考虑 DDoS 攻击了，不过这种情况是不可能存在的。因此，作为运维立场方面的考虑，我们不得不预防 DDoS 攻击。

DDoS 攻击的概念:

DDoS 的攻击方式有很多种，最基本的 DoS 攻击就是利用合理的服务请求来占用过多的服务资源，从而使服务器无法处理合法用户的指令。

DDoS 攻击手段是在传统的 DoS 攻击基础之上产生的一类攻击方式。单一的 DoS 攻击一般是采用一对一的方式，当被攻击目标 CPU 速度低、内存小或者网络带宽小等各项性能指标不高时，它的效果是明显的。随着计算机与网络技术的发展，计算机的处理能力迅速增长，内存大大增加，同时也出现了千兆级别的网络，这使得 DoS 攻击的困难程度加大了——目标对恶意攻击包的"消化能力"加强了不少，例如你的攻击软件每秒钟可以发送 3 000 个攻击包，但我的主机与网络带宽每秒钟可以处理 10 000 个攻击包，这样一来攻击就不会产生什么效果。

这时候分布式的拒绝服务攻击手段（DDoS）就应运而生了。当理解了 DDoS 攻击的话，它的原理就很简单。如果说计算机与网络的处理能力加大了 10 倍，用一台攻击机来攻击不再能起作用的话，攻击者使用 10 台攻击机同时攻击呢？用 100 台呢？DDoS 就是利用更多的傀儡机来发起进攻，以比从前更大的规模来进攻受害者。

高速广泛连接的网络给大家带来了方便，也为 DDoS 攻击创造了极为有利的条件。在低速网络时代时，黑客占领攻击用的傀儡机时，总是会优先考虑离目标网络距离近的机器，因为经过的路由器的跳数越少，效果就越好。而现在电信骨干节点之间的连接都是以 G 为级别的，大城市之间更可以达到 2.5G 的连接，这使得攻击可以从更远的地方或者其他城市发起，攻击者的傀儡机位置可以分布在更大的范围，选择起来更加灵活了。

—— 来自于互联网

21.1 Limit request 模块

在 Nginx 中通过使用限制连接数的方法可以起到阻止 DDoS 攻击的目的，由模块 Limit request 来实现，它提供了三个命令：limit_req_log_level、limit_req_zone 和 limit_req。

1. 实例

我们看下面的一个例子：

```
[root@dl ~]# cat  /usr/local/nginx0.8.53/conf/nginx.conf

…

http {
```

```
…
limit_req_log_level warn;
limit_req_zone  $binary_remote_addr  zone=ONLY_one:10m   rate=1r/s;
server {
listen   80;
server name  bbs.xxx.com;
limit_req   zone=ONLY_one  burst=5;
location / {
root   html;
index  index.html index.htm;
}
   location /download {

}
        …
}
```

2. 指令

在上面的实例中，limit_req_log_level、limit_req_zone 和 limit_req 这三个命令都用到了。下面认识一下这三个命令。

指令名称：limit_req_log_level

功能：该指令用于控制记录延时消息日志的级别，它的默认值为 warn，只能放置在 http 区段中。

指令名称：limit_req_zone

功能：该指令定义了一个区域，用于存储会话状态，至于会话中存储的是什么值，则由变量来决定。该指令有三个值，指定变量是第一个值，在这里我们使用的是 **$binary**_remote_**addr**，在上面的实例中指定存储会话的 zone 名字为“**ONLY_one**”，并且指定用于存储会话的空间为 10MB，这是第二个值，至于第三个值就是 **rate=1r/s**，它表示对于该 zone 的平均查询速度，单位是每秒钟多少个请求，在本例中限制了每秒钟一次请求(实际应用中可以将这个值设置为 10 左右，然后根据实际情况再进行调试)。设定了会话限制后，访问这里的每一个会话都将会被跟踪。另外在这里我们仍旧使用的是$binary_remote_addr 而不是$remote_addr，这是为了减少会话状态的字节数，使用二进制形式能够使得会话状态为 64 个字节，因此 1MB 的 zone 能够存储大约 16000 个会话状态。对于速度的单位，可以设置为：每秒的请求数（r/s）或每分钟的请求数（r/m），该指令只能放置在 http 区段中，它没有默认值。

指令名称：limit_req

功能：该指令指定的 zone—**ONLY_one**，并且同时指定了该 zone 最大可能的突发请求数（burst），如果请求的 **rate** 超过这个值，那么请求将会被延时，过量的请求被延时，直到请求的数量小于指定的 burst 值，因此对于请求率要有一个合适的速率。在这种情况下，如果你继续访问该 URL，将会以“Service unavailable”（503）结尾，注意，

默认的 burst 值为 0。此外，这个命令还有一个参数，那就是 nodelay，它的作用很简单，就是不延时处理。该指令可以放置在 http、server 和 location 区段中，它没有默认值。

受 DDoS 攻击的一般为博客、论坛和商业网站，例如，电子商务、企业网站等，因此一般针对这些网站做适当的限制。

21.2 访问测试

下面是一个针对使用了访问限制和没有使用访问限制的测试。

21.2.1 限制连接数

```
[root@bzd ~]# ab -n 1000 -c 100 http://192.168.3.150/download/
This is ApacheBench, Version 2.0.41-dev <$Revision: 1.141 $> apache-2.0
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Copyright(c)1998-2002TheApacheSoftwareFoundation, http://www.apache.org/

Benchmarking 192.168.3.139 (be patient)
Completed 100 requests
… //省略
Completed 900 requests
Finished 1000 requests

Server Software:nginx/0.8.53
Server Hostname:192.168.3.150
Server Port:80

Document Path:  /download/
Document Length:159 bytes

Concurrency Level:  100
Time taken for tests:   0.146372 seconds
Complete requests:  1000
Failed requests:1013
   (Connect: 0, Length: 1013, Exceptions: 0)
Write errors:   0
Non-2xx responses:  1040
Total transferred:  577570 bytes
HTML transferred:   398479 bytes
Requests per second:6831.91 [#/sec] (mean)
Time per request:   14.637 [ms] (mean)
Time per request:   0.146 [ms] (mean, across all concurrent requests)
Transfer rate:  3853.20 [Kbytes/sec] received
```

```
Connection Times (ms)
  min  mean[+/-sd] median   max
Connect:24   0.8  5   6
Processing: 47   1.7  8  13
Waiting:15   1.2  6  10
Total:  8   12   1.2 13  17
WARNING: The median and mean for the initial connection time are not within
a normal deviation
These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
  50% 13
… //省略
  99% 15
 100% 17 (longest request)
```

同时查看 Nginx 的访问日志（tail -f /usr/local/nginx0.8.53/logs/access.log）：

```
…
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
192.168.3.98 - - [12/Dec/2010:14:04:27 +0800] "GET /download/ HTTP/1.0" 503
383 "-" "ApacheBench/2.0.41-dev"
…
```

其中大量的 503 响应代码。

21.2.2 未限制连接数

我们再看一个没有做过限制连接数配置的测试：

```
[root@bzd ~]# ab -n 1000 -c 10 http://192.168.3.150/download/
…
Completed 900 requests
Finished 1000 requests

…
Concurrency Level:  10
```

```
Time taken for tests:   0.177107 seconds
Complete requests:  1000
Failed requests:0
Write errors:   0
…
Connection Times (ms)
  min  mean[+/-sd] median   max
Connect:00   0.0  0   0
Processing: 00   0.1  0   1
Waiting:00   0.0  0   0
Total:  00   0.1  0   1

Percentage of the requests served within a certain time (ms)
  50%  0
…
  99%  0
 100%  1 (longest request)
```

再看下面的三个例子。

例 1

配置如下：

```
http {
include   mime.types;
default_type  application/octet-stream;
sendfileon;
keepalive timeout  5;
limit req log level warn;
limit req zone  $binary remote addr  zone=one:10m   rate=5000r/s;
server {
listen   80;
server_name  localhost;
limit_req   zone=one  burst=5000;
location /download/ {
root   html;
index  index.html index.htm;
}
```

在进行“ab -n 1000 -c 100　http://192.168.3.175/download/”测试时，查看进程：

```
[root@kf ~]# lsof -i:80|grep nginx |wc -l
2
[root@kf ~]# lsof -i:80|grep nginx|wc -l
4
```

查看执行情况：

```
[root@bzd ~]# time  ab -n 1000 -c 100 http://192.168.3.175/download/
…
Finished 1000 requests

…

Concurrency Level:  100
Time taken for tests:  0.205198 seconds
Complete requests:  1000
Failed requests:0
Write errors:  0

…

  98% 21
  99% 21
 100% 22 (longest request)

real0m0.220s
user0m0.021s
sys 0m0.148s
```

例 2

配置如下：

```
http {
include  mime.types;
default type  application/octet-stream;
sendfileon;
keepalive_timeout  5;
limit_req_log_level warn;
limit_req_zone  $binary_remote_addr  zone=one:10m  rate=1r/s;
server {
listen  80;
server name  localhost;
limit_req  zone=one  burst=5  nodelay;
location /download/ {
root  html;
index  index.html index.htm;
}
```

在进行"ab -n 1000 -c 100 http://192.168.3.175/download/"测试时，查看进程：

```
[root@mail ~]# lsof -i:80|grep nginx|wc -l
2
```

执行 ab 测试，并查看报告：

```
[root@bzd ~]# time  ab -n 1000 -c 100 http://192.168.3.175/download/
…
Finished 1000 requests

…
Concurrency Level:  100
Time taken for tests:   0.139945 seconds
Complete requests:  1000
Failed requests:1001
   (Connect: 0, Length: 1001, Exceptions: 0)
Write errors:   0
Non-2xx responses:  1016
Total transferred:  566100 bytes
…
  99% 12
 100% 13 (longest request)

real0m0.154s
user0m0.033s
sys 0m0.114s
```

执行的速度很快，但是结果令人惊讶——“**Failed requests: 1001**”。

例 3

配置如下：

```
http {
include   mime.types;
default type  application/octet-stream;
sendfileon;
keepalive timeout  5;
limit_req_log_level warn;
limit_req_zone  $binary_remote_addr  zone=one:10m   rate=1r/s;
server {
listen   80;
server name  localhost;
limit req   zone=one  burst=5000;
location /download/ {
root   html;
index  index.html index.htm;
}
```

在进行“ab -n 1000 -c 100 http://192.168.3.175/download/”测试时，查看进程：

```
[root@mail ~]# lsof -i:80|grep nginx|wc -l
102
[root@mail ~]# lsof -i:80|grep nginx|wc -l
102
[root@mail ~]# lsof -i:80|grep nginx|wc -l
102
```

查看报告：

```
[root@bzd ~]# time  ab -n 1000 -c 100 http://192.168.3.175/download/

…
Finished 1000 requests

…

Concurrency Level:  100
Time taken for tests:   998.938111 seconds
Complete requests:  1000
Failed requests:0
Write errors:   0
…

Percentage of the requests served within a certain time (ms)
  50%  99993
…
 100%  99995 (longest request)

real16m38.950s
user0m0.026s
sys 0m0.304s
```

可以看得出，执行速度很慢，共用了 16 分 39 秒（=999 秒，绝对可以按照 1000 秒算！这将非常符合我们的设定——rate=1r/s）。

此外，预防 DDoS 攻击还可以通过限制 Nginx 的最高连接数、减少 keepalive_timeout 的值等措施，在这里就不再多说了。

第 22 章　为 Nginx 添加、清除或改写响应

本章包括了两个模块：HttpHeadersModule 和 ngx_headers_more，通过它们提供的指令来实现为 Nginx 添加、清除或者改写响应头。

22.1 HttpHeadersModule

通过 HttpHeadersModule 可以设置 HTTP 头，但是不能重写已经存在的头，比如可能相对 server 头进行重写，可以添加其他的头，例如 Cache-Control，设置生存期。

1. 配置示例

```
expires   24h;
expires   modified +24h;
expires   @15h30m;
expires   0;
expires   -1;
expires   epoch;
add_headerCache-Control  private;
```

2. 指令

HttpHeadersModule 提供了以下两条指令，即 add_header 和 expires。

指令名称：add_header

语法：add_header name value

默认值：none

使用环境：http，server，location

功能：为 HTTP 响应添加头。

> 注意，只有在响应代码为 200，204，301，302 或 304 时才有效。同样需要注意的是，除了 Last-Modified 头外，该指令可以在输出的头列表中添加一个新的头，但是不能使用这条指令来重写已经存在的头，比如可能相对 server 头进行重写（如果真想这么做，那么可以使用 HttpHeadersMoreModule）。

指令名称：expires

语法：expires [[modified] time|@time-of-day|epoch|max|off]

默认值：expires off

使用环境：http，server，location

功能：该指令用于控制是否在响应中添加一个生存期标志。

- off：阻止改变 Expires 和 Cache-Control 头。

- epoch：设置 Expires 头为 1 January，1970 00:00:01 GMT。
- max：设置 Expires 头为 31 December 2037 23:59:59 GMT，并且 Cache-Control 头的 max-age 值设为 10 年。
- [modified] time：如果设置一个不带@前缀的时间值，那么它表示页面的生存时间依赖于响应时间（如果这个时间值之前没有“modified”）或者是文件的修改时间（当“modified”存在，在 Nginx 0.7.0 和 0.6.32 版本中有效）。也可以设置为一个负值，这将会将 Cache-Control 头设置为 no-cache。
- @time-of-day：如果设置一个带有@前缀的时间值，那么缓存页面的计算方法是：“当前的时间”+“你指定的时间”。例如，“expires 24h”将会返回一个从现在开始向前推 24 小时的时间值，时间的格式为：Hh 或 Hh:Mm，这里 Hh 的值为 0～24，Mm 的值为 0～59（从 0.7.9 和 0.6.34 版本有效），h 和 m 表示单位。
- 如果设置为一个非负数或者是一个时间值，那么会将 Cache-Control 头的 max-age = #，即将 max-age 的值设置为“#”，这里的“#”将非负数或时间值转换成秒数。

3. 使用实例

实例 1

在 Nginx 的配置文件中添加以下配置内容：

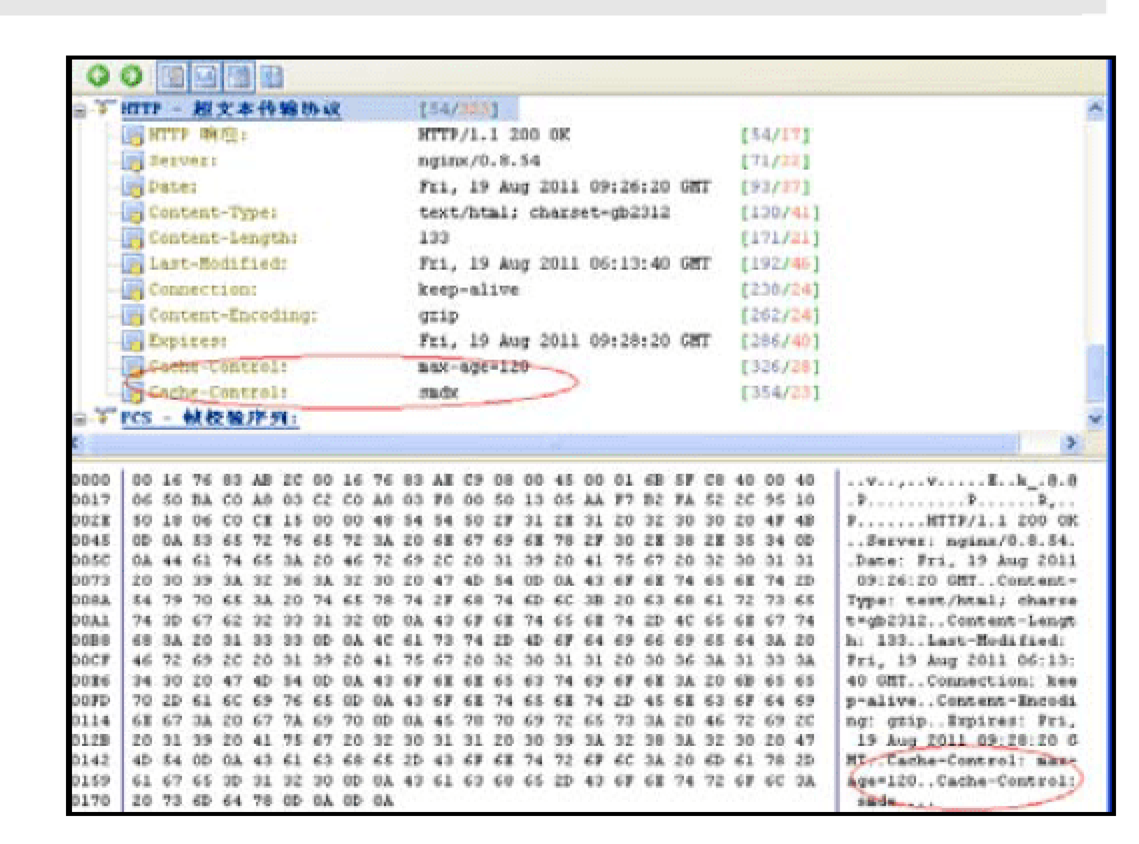
```
location / {
root    html;
index  index.html index.htm;
charset   gb2312;

expires   2m;
add_headerCache-Control  smdx;

gzip on;
gzip_types   text/plain application/xml;
gzip_static on;
 }
```

重新载入配置文件，然后再访问 http://www.xx.com：

我们看一下捕获的包，如右图所示：



注意圈起的部分，其中的两个头就是来自于我们在配置文件中的设置。max-age: 指定缓存过期的相对时间秒数。

实例 2

在 Nginx 的配置文件中添加以下配置：

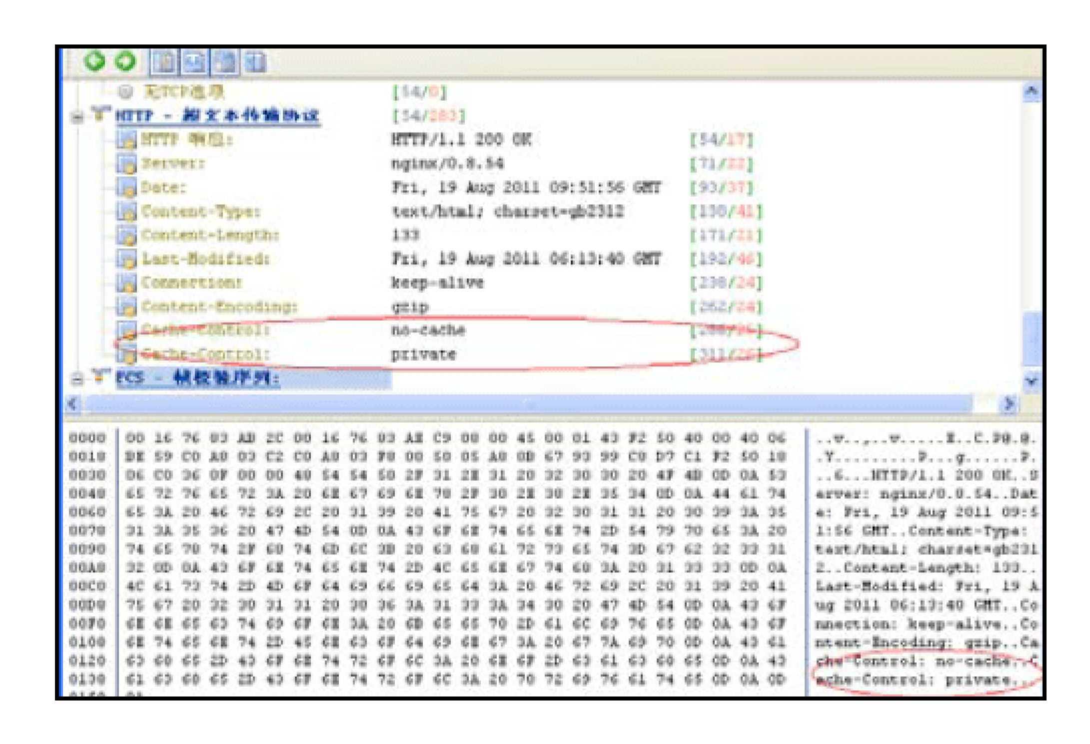
```
location / {
root   html;
index  index.html index.htm;
charset   gb2312;

add_header Cache-Control no-cache;
add_header Cache-Control private;

gzip on;
gzip_types   text/plain application/xml;
gzip_static on;
}
```

重新载入配置文件，然后再访问 http://www.xx.com：

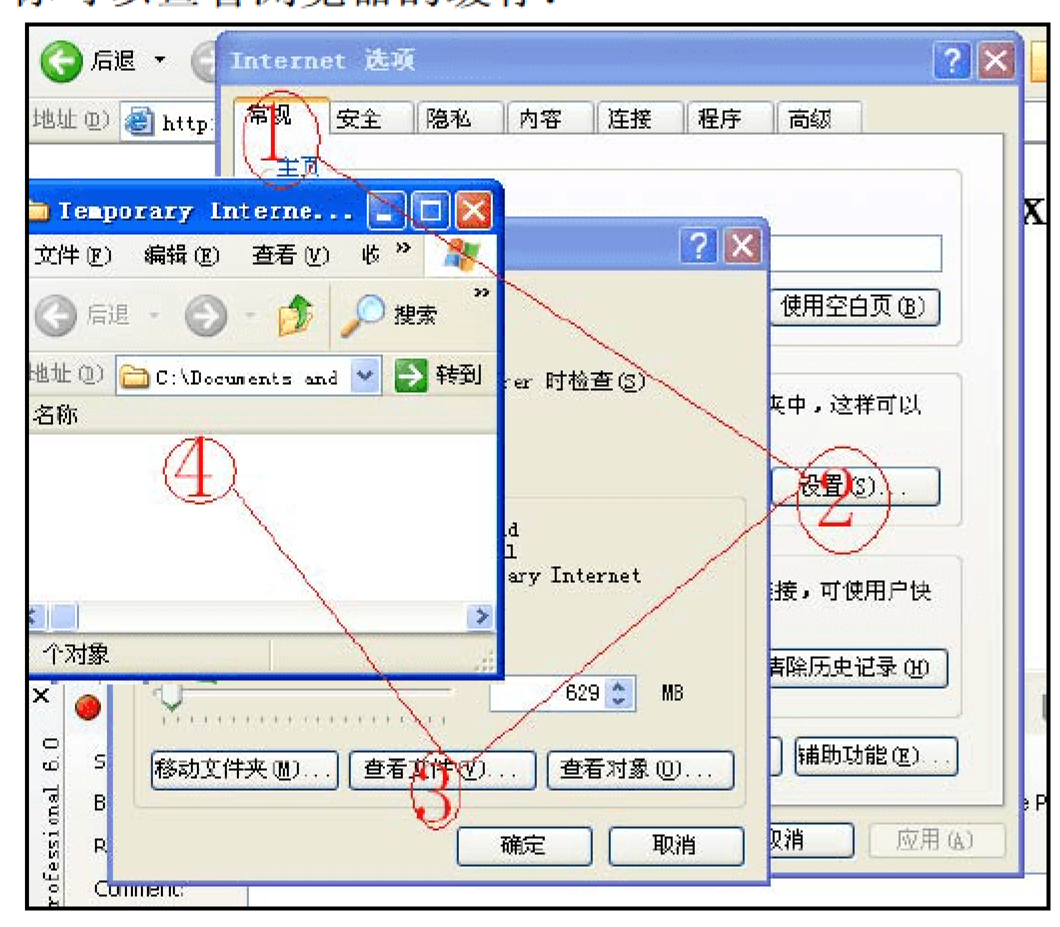
我们看一下捕获的包：



这次我们需要查看 Nginx 的访问日志，按 F5 键刷新网页：

```
192.168.3.248 - - [19/Aug/2011:18:01:01 +0800] "GET / HTTP/1.1" 200 133 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
```

你可以再刷新 N 次，都会发现日志中是“**GET / HTTP/1.1**”**200**，而不会出现“GET / HTTP/1.1”304，同时你可以查看浏览器的缓存：



缓存是空的！绝对是空的！

因此，如果不想让代理、缓存或者是浏览器缓存是空的，那么加 no-cache 参数或 private 参数。

实例 3

在 Nginx 的配置文件中添加以下配置：

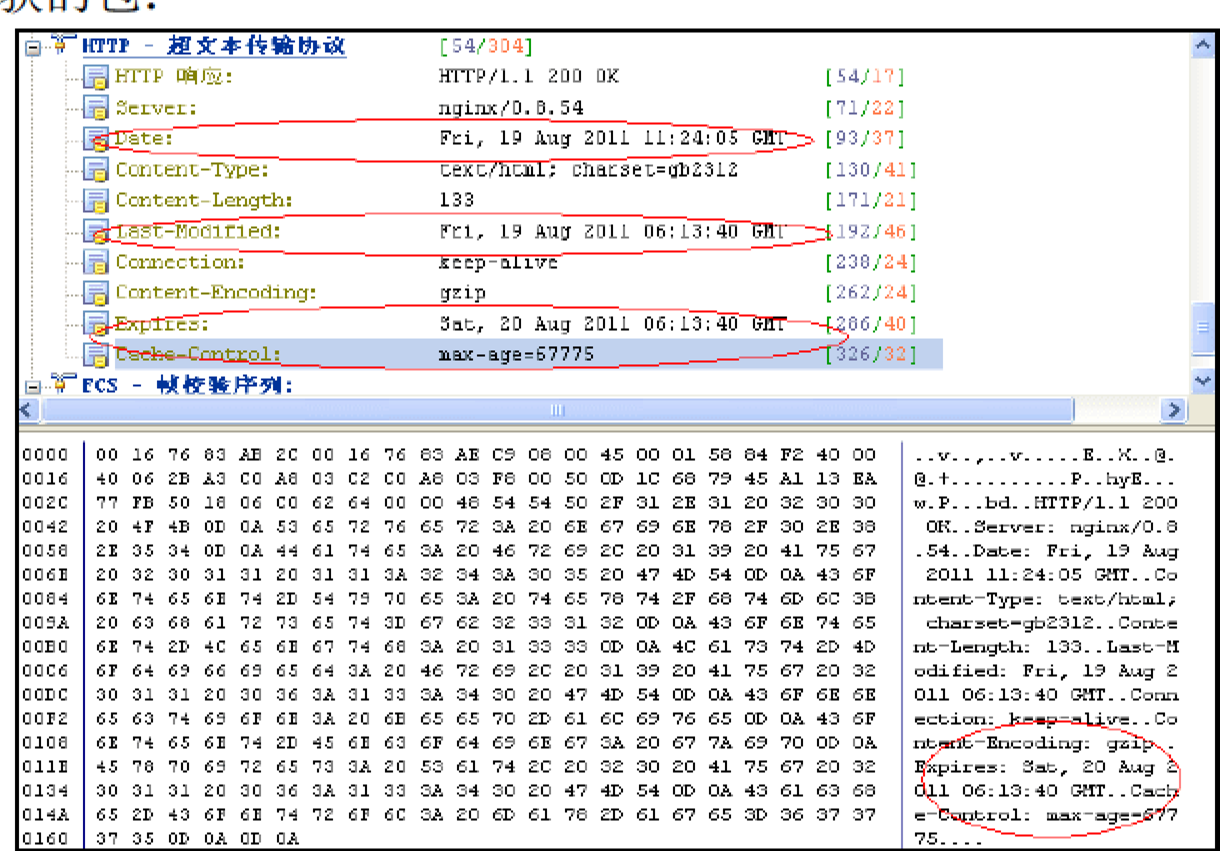
```
location / {
root   html;
index  index.html index.htm;
charset   gb2312;

expires   modified +24h;

gzip on;
gzip_types   text/plain application/xml;
gzip_static on;
}
```

重新载入配置文件，然后再访问 http://www.xx.com：

我们看一下捕获的包：



> 注意两个时间点和一个绝对值（这里用的是 GMT 时间——时差 8 小时）：

两个时间点是：Last-Modified 和 Expires。

```
Last-Modified:   Fri, 19 Aug 2011 11:24:05 GMT
Expires: Sat, 20 Aug 2011 06:13:40 GMT
Cache-Control:   max-age=67775
```

说明一下：Last-Modified，Nginx 服务器页面的最后修改时间，例如，文件的最后修改时间、动态页面的最后产生时间。例如：

```
[root@mail html]# stat 50x.html
  File: '50x.html'
  Size: 383 Blocks: 8  IO Block: 4096   regular file
Device: fd00h/64768dInode: 294859  Links: 1
```

```
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-08-18 14:16:34.000000000 +0800
Modify: 2011-08-16 10:12:00.000000000 +0800
Change: 2011-08-16 10:12:00.000000000 +0800
```

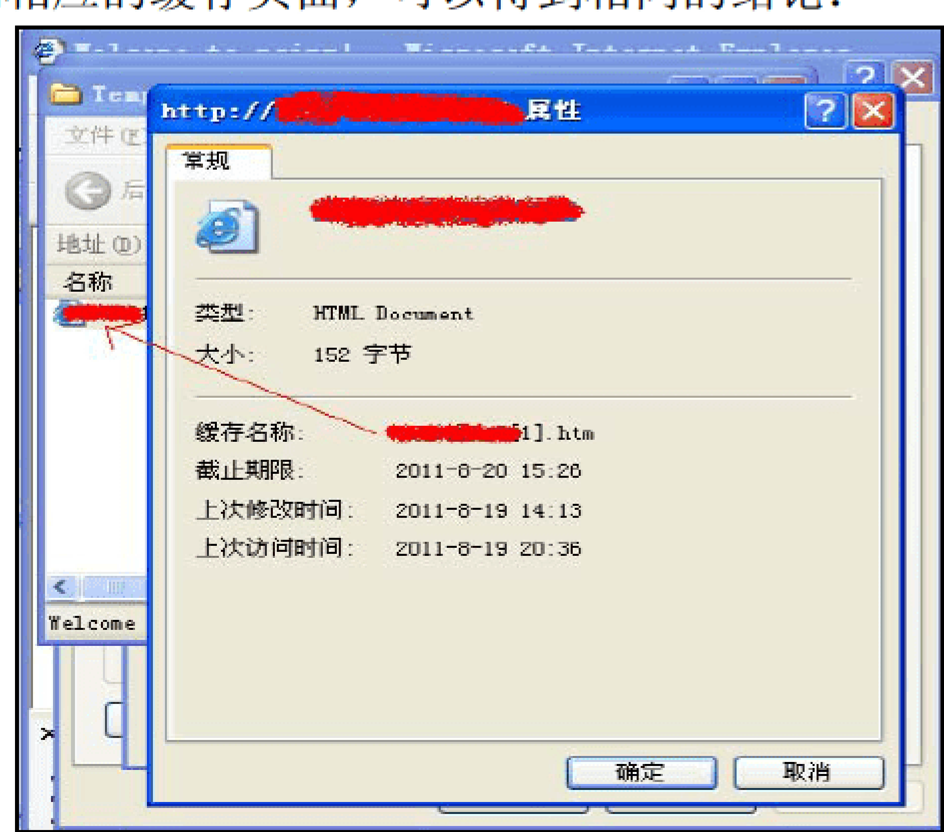
就是黑体字标出部分的时间。

我们比较一下 Last-Modified 和 Expires 的时间差：为 18 个小时 49 分 35 秒。那么就是说，如果客户端不清除缓存，在这个时间段内，无论何时访问该网页，访问的都是本地浏览器缓存中的文件。

一个绝对值是 Cache-Control，至于为什么是 Cache-Control: max-age=67775，而不是 86400（24 小时）？这其实很容易理解：

我们算过 Last-Modified 和 Last-Modified 的时间间隔是：18 小时 49 分 35 秒，只要把它折算成秒值，那么就是 max-age 的值：18*3600+49*60+35=67775。

从客户端缓存中找到相应的缓存页面，可以得到相同的结论：



可以通过“截止期限”和“上次访问时间”来计算，那么得出的值为：18 小时 50 分，也就是我们在前面折算出的 max-age 值，只不过在 Windows 下没有精确到秒。也就是说，如果客户端不清除缓存，在这个时间段内，无论何时访问该网页，访问的都是本地浏览器缓存中的文件。

因此，无论是从服务器端响应包角度分析还是从客户端缓存文件角度分析，结果是一致的。

参考以下这个例子：

缓存期没过时的访问：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Server: nginx/0.8.54
Date:  Fri, 19 Aug 2011 10:58:37 GMT //客户端的访问时间
```

```
Content-Type: text/html; charset=gb2312
Content-Length: 152
Last-Modified: Thu, 18 Aug 2011 11:07:54 GMT
Connection: keep-alive
Expires:  Fri, 19 Aug 2011 11:07:54 GMT
Cache-Control: max-age=557
Accept-Ranges: bytes

 …//网页内容省略

Connection closed by foreign host.
```

缓存期已过时的访问：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4...
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Server: nginx/0.8.54
Date:  Fri, 19 Aug 2011 11:08:41 GMT //客户端的访问时间
Content-Type: text/html; charset=gb2312
Content-Length: 152
Last-Modified: Thu, 18 Aug 2011 11:07:54 GMT
Connection: keep-alive
Expires:  Fri, 19 Aug 2011 11:07:54 GMT
Cache-Control: no-cache //控制缓存的时间为 0，就变成了 no-cache
Accept-Ranges: bytes

 …//网页内容省略

Connection closed by foreign host.
```

4. Cache-Control 头

从上面的这些例子中我们了解到，页面在缓存中（缓存服务器或者是客户端浏览器缓存）缓存的时间由“Cache-Control”头来控制的。对于它的取值，可以从两方面来分析，即请求时的缓存指令和响应消息中的指令。

- 请求时的缓存指令：no-cache、no-store、max-age、max-stale、min-fresh、only-if-cached
- 响应消息中的指令：public、private、no-cache、 no-store、no-transform、must-revalidate、proxy-revalidate、s-maxage

我们看一下这些指令。

- no-cache：该指令表示请求或响应消息将不会缓存。它会强制浏览器在使用缓存复制之前先提交一个 http 请求到服务器进行确认。
- no-store：如果设置为该值，那么无论是远程还是本地，是共享还是非共享的缓存，都不能存储访问页面的缓存副本，这样有效地预防了一些重要信息的发布，如果在请求中发送了该指令，那么将会使得请求和响应中都不会使用缓存。换句话说就是，浏览器在任何情况下都不要进行数据缓存，不在本地保留复制。
- max-age：该参数的值就是客户端可以在缓存中保存缓存页面的最大时间，单位为秒。
- max-stale：如果指定了该值，那么客户端可以接收超出最大生存期的响应，否则，对于超出最大生存期的响应，客户端将不会接收。如果为该参数设定了值，那么表示只在该值范围内才可以提供数据访问。
- min-fresh：表示客户端可以接收响应时间小于当前时间加该参数上指定时间的响应时间。
- only-if-cached：设置为只读取缓存。（Valid document was not found in the cache and only-if-cached directive was specified.——这种错误就是由于设置了 only-if-cached 引发的，由于在本地缓存中没有找到缓存，就报这种错误了）
- public：设定响应数据可以被任何缓存区缓存。
- private：表示私有数据不能缓存。
- no-transform：如果使用该值，那么将禁止下游代理服务器更改 Content-Encoding、Content-Range 或 Content-Type 头指定的任何标头值，包括页面本身内容。
- must-revalidate：如果设置为该值，那么就会强制浏览器重新验证文件是否过期，如果不加，有些浏览器还是会保留部分缓存。
- proxy-revalidate：如果设置该值则会强制 proxy 严格遵守在 Nginx 服务器端设置的缓存规则。
- s-maxage：功能类似于 max-age，但是它只用在共享缓存上，比如 proxy。

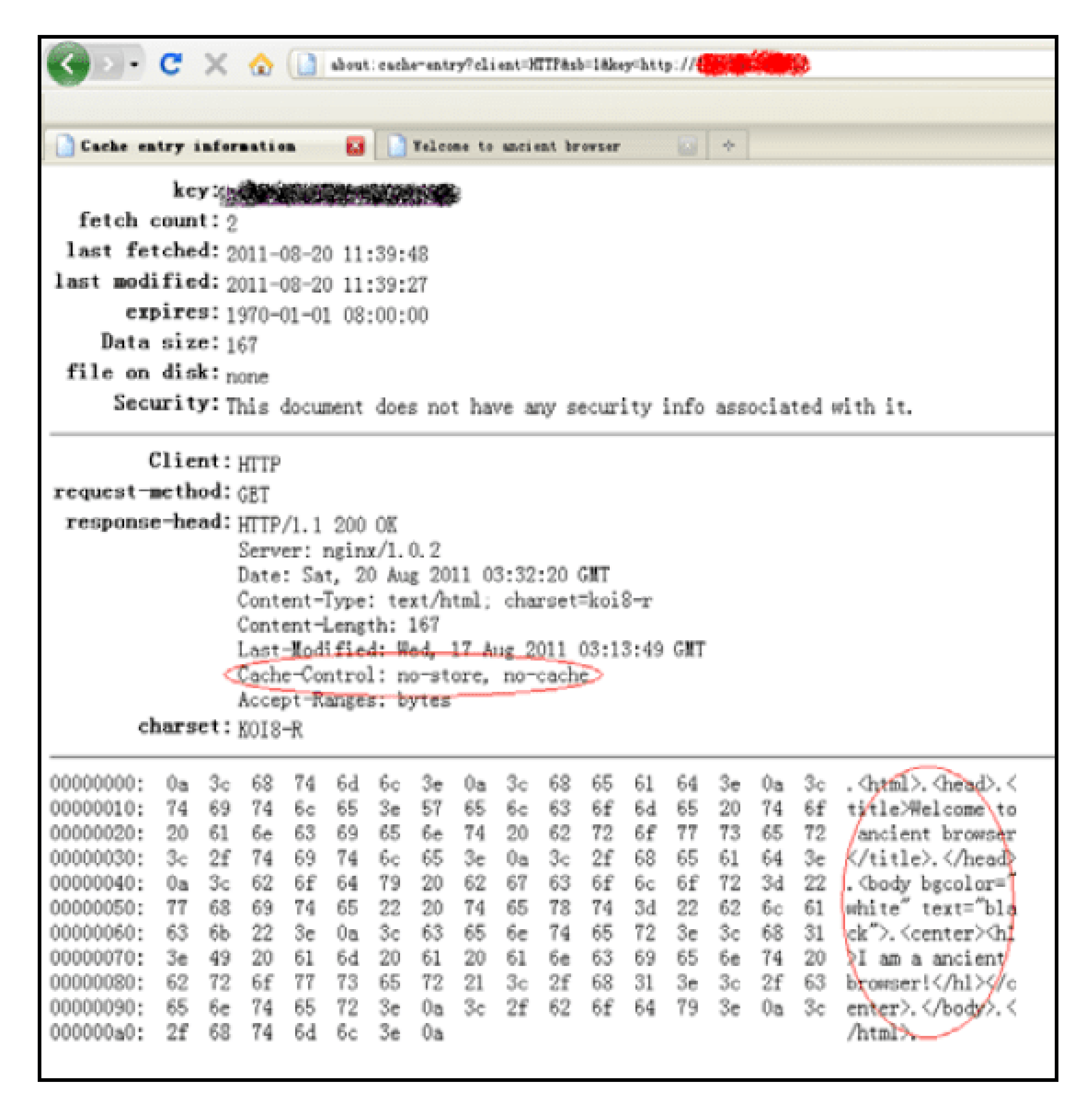
在 Nginx 中添加以下配置：

```
location / {
  root   html;
  index  index.html index.htm;

  add_header Cache-Control no-store;
  add_header Cache-Control no-cache;

}
```

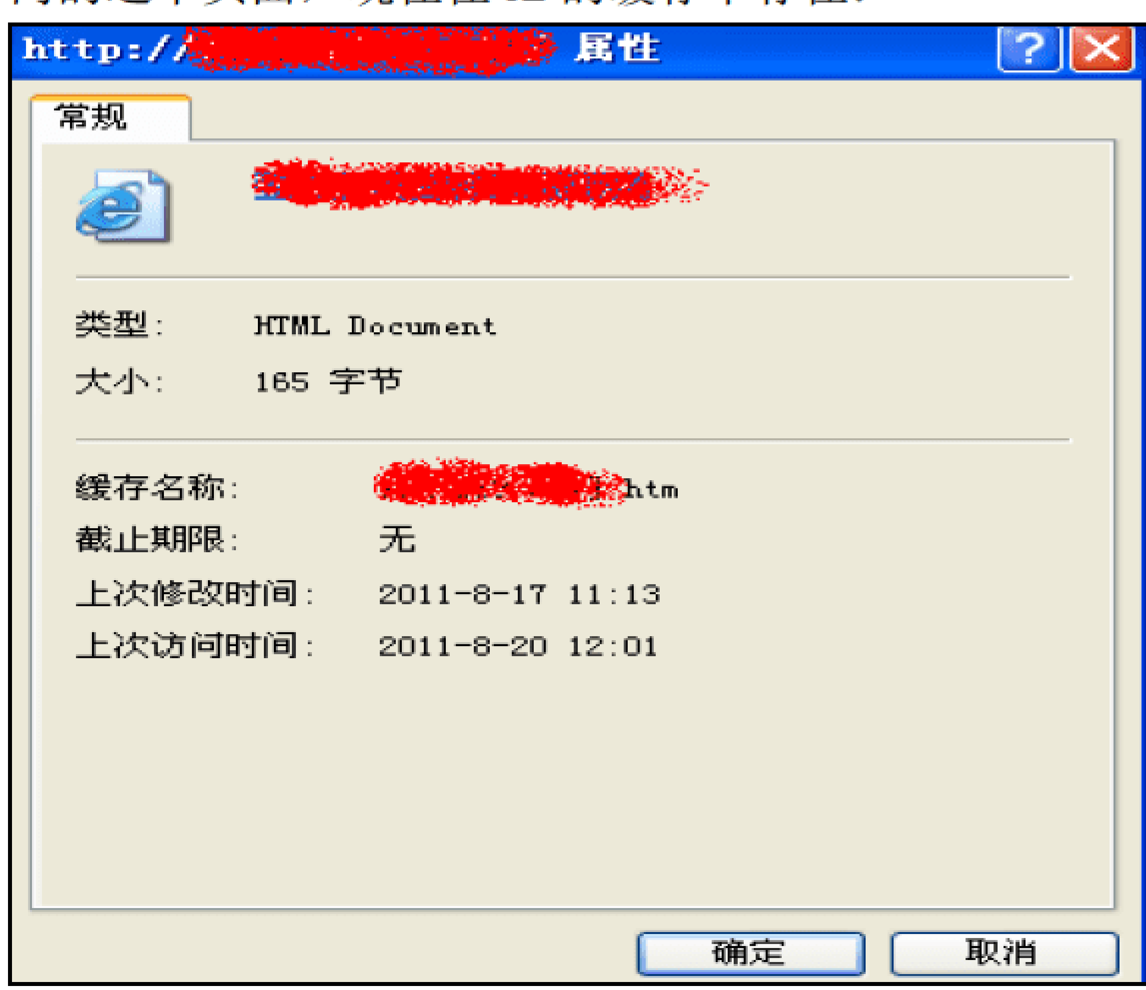
我们使用 firefox 浏览器来访问 http://www.xx.com，查看它的缓存：



很明显，缓存中存在该页面。但是每次刷新页面，Nginx 的访问日志中都是 200，没有 304，也就是说从服务器端传输数据，而不会去读缓存，但是每次请求获取的响应数据，也就是我们获取的页面又会被缓存。

在这种情况下，如果使用 IE 浏览器访问则不会有缓存。但是 IE 浏览器会有这样的情况：

假设我们将要访问的这个页面，现在在 IE 的缓存中存在：



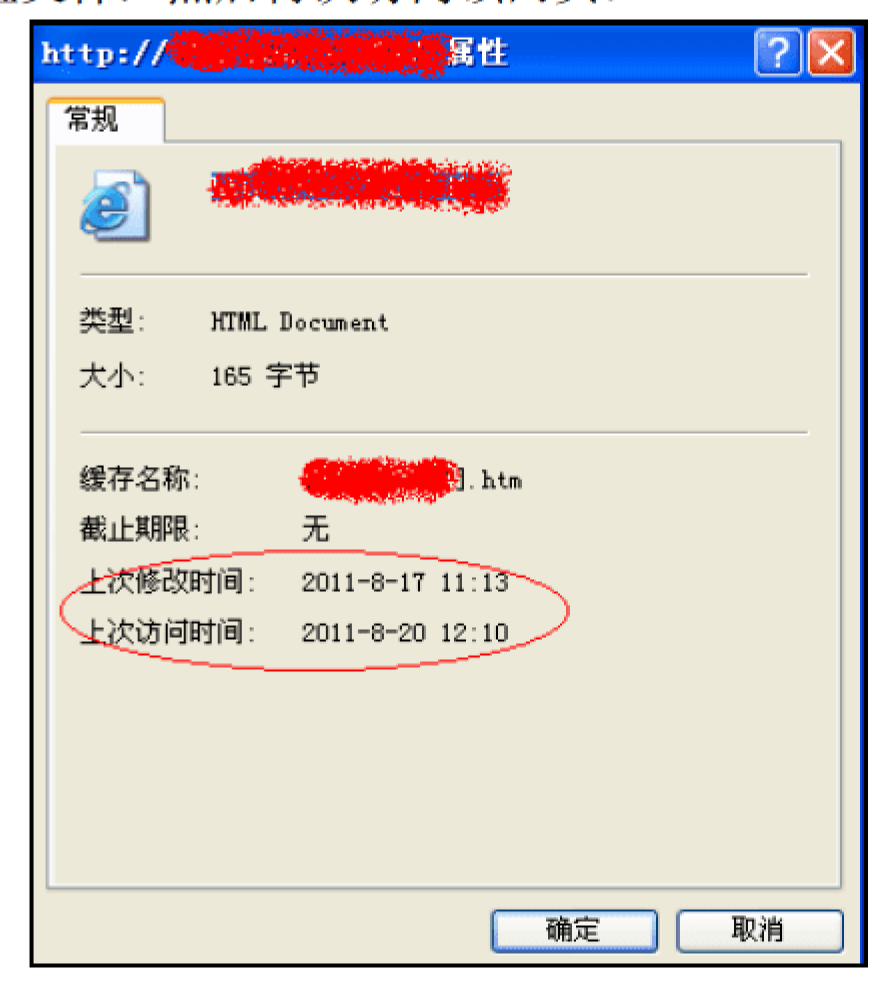
注意图上的时间，在图中没有圈出。

在 Nginx 的配置中添加以下配置：

```
location / {
  root   html;
  index  index.html index.htm;
```

```
    add_header Cache-Control no-store;
 }
```

重新载入 Nginx 的配置文件，然后再次访问该网页：



我们清楚地看到，在 IE 缓存中存储的数据被访问了，而且在 Nginx 的访问日志中也是 304，而不是 200。

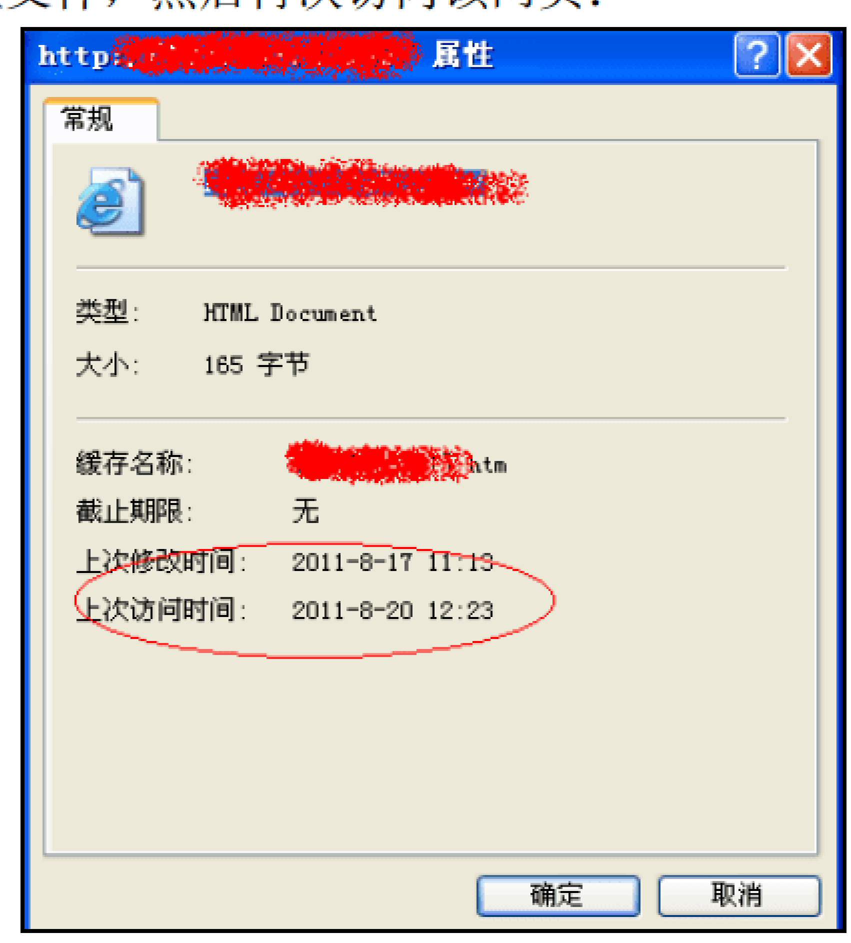
如果在 Nginx 的配置中添加以下配置：

```
 location / {
    root   html;
    index  index.html index.htm;

    add_header Cache-Control no-cache;

 }
```

重新载入 Nginx 的配置文件，然后再次访问该网页：



同样，我们清楚地看到，在 IE 缓存中存储的数据被访问了，而且在 Nginx 的访问日志中也

是 304，而不是 200。

其他的浏览器我们就没必要测试了，强调的就是，如果从数据安全角度来看待这个问题，那么这两个参数都是做不到的。

22.2 ngx_headers_more

ngx_headers_more 模块可以用来设置、清除进入以及出去的头，并且可以添加其他头。在默认安装时没有安装该模块，是一个第三方模块，因此我们需要下载并且安装后才可以使用。

ngx_headers_more 模块是我们刚刚在前面认识的 header 模块的增强版，因为它能够提供更多的功能，例如，重新设置和清除“builtin headers”，即内置的各种头，如 Content-Type、Content-Length 和 Server。

在使用 more_set_headers 和 more_clear_headers 修改输出头时，也可以通过该模块的-s 选项来指定一个可选的 HTTP 状态代码，使用-t 选项来指定一个 Content-Type。例如：

```
more_set_headers -s 404 -t 'text/html' 'X-Foo: Bar';
```

输入头（就是客户端的请求头）也可被修改，例如：

```
location /foo {
   more_set_input_headers 'Host: foo' 'User-Agent: faked';
   # now $host, $http_host, $user_agent, and
   #   $http_user_agent all have their new values.
}
```

选项 -t 在指令 more_set_input_headers 和 more_clear_input_headers 中都可以使用，同样有效，但是选项-s 却不可以。

不像我们在前面使用的标准 header 模块，该模块的指令默认应用所有的状态代码，包括 4xx 和 5xx。

模块的兼容性

下列是 Nginx 的版本和该模块的版本之间的匹配情况：

- 1.0.x （last tested: 1.0.5）
- 0.9.x （last tested: 0.9.4）
- 0.8.x （last tested: 0.8.54）
- 0.7.x >= 0.7.44 （last tested: 0.7.68）

Nginx 早期的版本，例如 0.6.x 和 0.5.x 都不能使用该模块。

1. 安装 headers-more

下载并且安装 headers-more 模块。

下载 headers-more：

```
[root@mail ~]# wget https://nodeload.github.com/agentzh/  \
> headers-more-nginx-module/zipball/v0.15rc3
[root@mail ~]# unzip agentzh-headers-more-nginx-module-v0.15rc3-0
-g5fac223.zip
```

安装 Nginx：

```
[root@mail nginx-1.0.2]# ./configure\
> --prefix=/usr/local/nginx-1.0.2-headers-more
> --add-module=/root/agentzh-headers-more-nginx-module-5fac223/
[root@mail nginx-1.0.2]#make -j2
[root@mail nginx-1.0.2]#make install
```

2. 配置示例

```
# 设置 Server 输出头
more_set_headers 'Server: my-server';

# 设置和清除输出头
location /bar {
   more_set_headers 'X-MyHeader: blah' 'X-MyHeader2: foo';
   more_set_headers -t 'text/plain text/css' 'Content-Type: text/foo';
   more_set_headers -s '400 404 500 503' -s 413 'Foo: Bar';
   more_clear_headers 'Transfer-Encoding' 'Content-Type';

   # 将 proxy_pass/memcached_pass/或者是其他配置放置在这里...
}

# 设置输出头
location /type {
   more_set_headers 'Content-Type: text/plain';
   # …
}

# 设置输入头
location /foo {
   set $my_host 'my dog';
   more_set_input_headers 'Host: $my_host';
   more_set_input_headers -t 'text/plain' 'X-Foo: bah';

   #从现在开始$host 和 $http_host 的值都会使用新设置的值...
   # …
}

# 替代 X-Foo 输入头 ，仅在该头存在的情况下才会有替代行为发生
more_set_input_headers -r 'X-Foo: howdy';
```

3. 指令

ndx_headers_more 模块提供了四条指令，下面我们来看一下它们的用法。

指令名称：more_set_headers

语法：more_set_headers [-t <content-type list>]... [-s <status-code list>]... <new-header>...

默认值：no

使用环境：http，server，location，location 中 if 区段

功能：当响应代码匹配通过-s 选项指定的代码，并且响应内容类型匹配通过-t 选项指定的类型，那么添加或者取代指定的输出头。有关 content-type 列表可取的值见后面部分。

如果-s 或者-t 选项有一个没有指定，或者是它们的值为空，都不是匹配所必须的，因此，下列指令设置的 Server 输出头为自定义值“my_server”，没有指定-t 和-s，则表示匹配任何状态代码和任何内容类型：

```
more_set_headers"Server: my_server";
```

单个指令可以设置、添加多个输出头。例如：

```
more_set_headers 'Foo: bar' 'Baz: bah';
```

如果在一条指令中多次出现了同一个选项，这也是可以的，但在执行时它们会被合并到一起。例如：

```
more_set_headers -s 404 -s '500 503' 'Foo: bar';
```

这条指令相当于：

```
more_set_headers -s '404 500 503' 'Foo: bar';
```

新添加的头可以是以下格式之一：

- Name：Value
- Name：
- Name：

最后两种实际上是清除了头 Name 的值，我们会在后面的例子中看到。

Nginx 的变量能够用在头的值中，例如：

```
set  $my_var "dog";
more_set_headers "Server: $my_var";
```

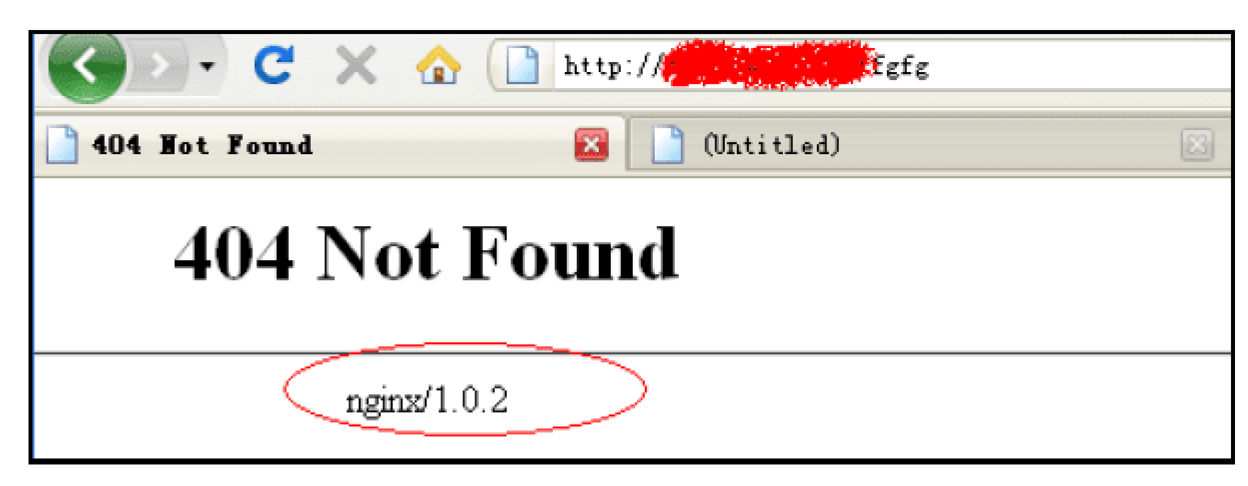
但是出于性能方面的考虑，不要将变量引入头的值中。

如果在 Nginx 的配置文件中添加了这样的配置，那么看一下访问的结果：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying www.xx.com...
Connected to www.xx.com （1.2.3.4）.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sat, 20 Aug 2011 10:34:39 GMT
Content-Type: text/html
Content-Length: 151
Last-Modified: Sat, 20 Aug 2011 05:50:04 GMT
Connection: keep-alive
Server: dog
```

如果不想暴露服务器的某些信息，可以在 response 头信息中自定义头，当然要是单纯地为了这个就没必要添加该模块了，我们在相关的章节介绍过，可以修改 Nginx 的源码后再进行安装。另外，这种隐含 Server 的方法对于以下这种情况还是没解决：



在同一个 location 中可以同时使用多个设置，清除头指令它们都能够被使用，都会被顺序执行，这个现象可以在后面的例子中看到。例如，我们在 Nginx 的配置文件中添加以下配置：

```
    location / {
  root   html;
  index  index.html index.htm;

more_set_headers 'Foo: bar' 'Baz: bah';
more_clear_headers -s 404 -t 'text/html' Foo Baz;
more_set_headers "Server:Microsoft-IIS/6.0";

}
```

那么我们进行以下访问：

首先访问一个不存在的目录，xxx 是一个不存在的目录，访问结果如下：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying www.xx.com...
Connected to www.xx.com (1.2.2.4).
Escape character is '^]'.
GET /xxx  HTTP/1.1
Host: www.xx.com

HTTP/1.1 404 Not Found
Date: Sat, 20 Aug 2011 12:13:26 GMT
Content-Type: text/html
Content-Length: 168
Connection: keep-alive
Server: Microsoft-IIS/6.0

… //省略

Connection closed by foreign host.
```

其次，再访问一个存在的目录，访问结果如下：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying www.xx.com...
Connected to www.xx.com (1.2.2.4).
Escape character is '^]'.
GET / HTTP/1.1
```

```
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sat, 20 Aug 2011 12:13:04 GMT
Content-Type: text/html
Content-Length: 151
Last-Modified: Sat, 20 Aug 2011 05:50:04 GMT
Connection: keep-alive
Foo: bar
Baz: bah
Server: Microsoft-IIS/6.0
Accept-Ranges: bytes

… //省略

Connection closed by foreign host.
```

看服务器端发回响应中的黑体字就明白了。

more_set_headers 指令会从上一级继承得到（就是说从 http 区段或者是 server 区段）而执行，而在 server 与 server、location 与 location 区段之间不会发生继承和替代。看下面的两个例子：

例 1

我们在 Nginx 的配置文件中添加以下配置：

```
http {
include   mime.types;
default type  application/octet-stream;
more_set_headers 'Foo: bar' 'Baz: bah';
sendfileon;
keepalive_timeout  65;

server {
listen   80;
server name  localhost;
   more_clear_headers -s 404 -t 'text/html' Foo Baz;
   more_set_headers "Server:Microsoft-IIS/6.0";

location /rmb {
root   html;
index  index.html index.htm;
more_set_headers 'RMB: bw';

}
 location /rmc {
root   html;
index  index.html index.htm;
```

```
    more_set_headers 'RMc: cw';

    }

 …  //省略
    }
```

访问**/rmb/index.html** 页面：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4...
Connected to www.xx.com (1.2.2.4).
Escape character is '^]'.
GET /rmb/index.html HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sat, 20 Aug 2011 12:27:04 GMT
Content-Type: text/html
Content-Length: 34
Last-Modified: Sat, 20 Aug 2011 12:23:29 GMT
Connection: keep-alive
Foo: bar
Baz: bah
Server: Microsoft-IIS/6.0
RMB: bw
Accept-Ranges: bytes

… //省略

Connection closed by foreign host.
```

访问**/rmc/index.html** 页面：

```
 [root@jh-share csgz]# telnet app1.xx.com 80
Trying 1.2.3.4...
Connected to app1.xx.com (1.2.3.4).
Escape character is '^]'.
GET /rmc/index.html HTTP/1.1
Host: app1.xx.com

HTTP/1.1 200 OK
Date: Sun, 21 Aug 2011 01:31:24 GMT
Content-Type: text/html
Content-Length: 34
Last-Modified: Sun, 21 Aug 2011 01:28:01 GMT
Connection: keep-alive
```

```
Foo: bar
Baz: bah
Server: Microsoft-IIS/6.0
RMC: cw
Accept-Ranges: bytes

…  //省略

Connection closed by foreign host
```

比较配置文件和两个访问页面的响应，看黑体字部分。通过这个例子说明了在 location 与 location 之间的配置不会发生继承和替代。

例 2

我们在 Nginx 的配置文件中添加以下配置：

```
http {
include   mime.types;
default type  application/octet-stream;
more_set_headers 'Foo: bar' 'Baz: bah';
sendfileon;
keepalive_timeout  65;

server {
listen   80;
server name  www.xx.com;
more_set_headers 'RMB: bw';
    more_clear_headers -s 404 -t 'text/html' Foo Baz;
    more_set_headers "Server:Microsoft-IIS/6.0";

location /rmb {
root   html;
index  index.html index.htm;

}
}

   server {
listen   80;
server name  app1.xx.com;
    more_set_headers 'RMJ: sw';
    more_clear_headers -s 404 -t 'text/html' Foo Baz;
    more_set_headers "Server: Apache/2.2.3 ";

location /rmb {
root   html;
```

```
    index  index.html index.htm;

    }

…  //省略
    }
```

访问 http://www.xx.com/rmb/index.html 页面：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.2.4).
Escape character is '^]'.
GET /rmb/index.html HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sat, 20 Aug 2011 12:27:04 GMT
Content-Type: text/html
Content-Length: 7
Last-Modified: Sat, 20 Aug 2011 12:23:29 GMT
Connection: keep-alive
Foo: bar
Baz: bah
RMB: bw
Server: Microsoft-IIS/6.0
Accept-Ranges: bytes

…  //省略

Connection closed by foreign host.
```

访问 http://app1.xx.com/rmj/index.html 页面：

```
[root@jh-share csgz]# telnet app1.xx.com 80
Trying 1.2.3.4…
Connected to app1.xx.com (1.2.3.4).
Escape character is '^]'.
GET /rmb/index.html HTTP/1.1
Host: app1.xx.com

HTTP/1.1 200 OK
Date: Sun, 21 Aug 2011 01:03:27 GMT
Content-Type: text/html
Content-Length: 33
Last-Modified: Sun, 21 Aug 2011 01:00:36 GMT
Connection: keep-alive
```

```
Foo: bar
Baz: bah
RMJ: sw
Server: Apache/2.2.3
Accept-Ranges: bytes

Hello,this is RMB on www.xx.com!

Connection closed by foreign host.
```

比较配置文件和两个访问页面的响应，看黑体字部分。通过这个例子说明了在 server 与 server 之间的配置不会发生继承和替代。

> 注意，尽管 more_set_headers 能够用在 location 区段中的 if 区段，但是不要用在 server 区段的 if 区段中，下列配置是不允许的:

```
# This is NOT allowed!
server {
if ($args ~ 'download') {
   more set headers 'Foo: Bar';
   }
 ...
}
```

指令名称：more_clear_headers

语法： more_clear_headers [-t <content-type list>]... [-s <status-code list>]... <header>...

默认值：no

使用环境：http，server，location，location 中 if 区段

功能：清除指定的输出头。

> 实际上，more_clear_headers -s 404 -t 'text/plain' Foo Baz 完全等同于 more_set_headers -s 404 -t 'text/plain' "Foo: " "Baz: "或者 more_set_headers -s 404 -t 'text/plain' Foo Baz。

通配符 * 也可以使用在头匹配中。例如，下面的指令有效地清除了任何以“X-Hidden-”开始的输出头：

```
more_clear_headers 'X-Hidden-*';
```

下面我们看一下该指令的用法，在 Nginx 的配置文件中添加以下配置：

例 1

```
 location / {
root   html;
index  index.html index.htm;
more_set_headers 'Foo: bar' 'Baz: bah';
more_clear_headers -s 404 -t 'text/html' Foo Baz;
```

```
more_set_headers "Server:Microsoft-IIS/6.0";

 }
```

访问一个存在的页面：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sun, 21 Aug 2011 03:10:58 GMT
Content-Type: text/html
Content-Length: 151
Last-Modified: Sat, 20 Aug 2011 05:50:04 GMT
Connection: keep-alive
Foo: bar
Baz: bah
Server: Microsoft-IIS/6.0
Accept-Ranges: bytes
```

访问一个不存在的页面：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET /xxx HTTP/1.1
Host: www.xx.com

HTTP/1.1 404 Not Found
Date: Sun, 21 Aug 2011 03:11:55 GMT
Content-Type: text/html
Content-Length: 168
Connection: keep-alive
Server: Microsoft-IIS/6.0
```

比较这两个访问的不同点：在第一个访问中由于页面的存在，因此，指令“**more_clear_headers -s 404 -t 'text/html' Foo Baz;**”没有起作用。而在第二个访问中，由于访问了一个不存在的页面，因此服务器端给客户端返回的响应代码为 404，所以这条指令就起作用了，故而在返回的响应头中没有 **Foo: bar**、**Baz: bah** 头了。

例 2

```
location / {
```

```
root   html;
index  index.html index.htm;
more_clear_headers -s 404 -t 'text/html' Foo Baz;
more_set_headers 'Foo: bar' 'Baz: bah';
more_set_headers "Server:Microsoft-IIS/6.0";

  }
```

同样访问一个不存在的页面，我们会看到以下结果：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 192.168.3.139…
Connected to www.xx.com (1.2.3.4) .
Escape character is '^]'.
GET /xxx HTTP/1.1
Host: www.xx.com

HTTP/1.1 404 Not Found
Date: Sun, 21 Aug 2011 09:32:51 GMT
Content-Type: text/html
Content-Length: 168
Connection: keep-alive
Foo: bar
Baz: bah
Server: Microsoft-IIS/6.0
```

如果出现这种现象，你可千万别感到意外。因为我在前面说过，这些指令是被顺序执行的，因此，在执行“**more_clear_headers -s 404 -t 'text/html'Foo Baz;**”指令时，并不是最终的访问，随后的指令同样会被执行，就出现了这种现象。在具体使用时要注意这一点。

指令名称：more_set_input_headers

语法：more_set_input_headers [-r] [-t <content-type list>]... <new-header>...

默认值：no

使用环境：http，server，location，location 中 if 区段

功能：该指令除了它是针对进入的头（或者叫客户端的请求头）以外，十分像 more_set_headers 指令，只是它只支持-t 选项。注意，该指令总是运行在 rewrite 指令结尾，以便在标准的 rewrite 模块之后运行，并使用在子请求中。

如果指定了-r 选项，那么就会发生替代，将原有的值替换为新的值，但前提条件是被替换的对象存在。

我们看以下这个例子：

```
 location / {
   root   html;
   index  index.html index.htm;

set $my_host 'my dog';
```

```
more_set_input_headers 'Host: $my_host';

more_set_headers"Server: $host";
 }
```

在这个例子中，我首先通过指令“**more_set_input_headers 'Host: $my_host';**”为客户端请求添加了一个请求头，然后再通过指令“**more_set_headers"Server: $host";**”将请求头发给客户端。我们看一下这一过程在 Nginx 执行中是否能实现：

```
server {
  listen   80;
  server name  localhost;

location /rmb {
  root   /www/rmb;
  index  index.html index.htm;

  set $c host 'RMB';
  more_set_input_headers 'Host: $c_host';

  more set headers"Server: $host";


}
location /rmc {
  root   /www/rmc;
  index  index.html index.htm;
  set $c host 'RMC';
  more_set_input_headers 'Host: $c_host';

  more_set_headers"Server: $host";


}
}
```

访问**/rmb/index.html**：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to appl.xx.com (1.2.3.4) .
Escape character is '^]'.
GET /rmb/index.html HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sun, 21 Aug 2011 10:53:08 GMT
```

```
Content-Type: text/html
Content-Length: 33
Last-Modified: Sun, 21 Aug 2011 01:00:36 GMT
Connection: keep-alive
Server: RMB
Accept-Ranges: bytes
```

访问 **/rmc/index.html**：

```
[root@jh-share csgz]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to app1.xx.com (1.2.3.4).
Escape character is '^]'.
GET /rmc/index.html HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Date: Sun, 21 Aug 2011 10:53:38 GMT
Content-Type: text/html
Content-Length: 34
Last-Modified: Sun, 21 Aug 2011 01:28:01 GMT
Connection: keep-alive
Server: RMC
Accept-Ranges: bytes
```

可见确实 **$host** 能被替换，这说明，当设置了指令“more_set_input_headers 'Host: $my_host';”以后，在 Nginx 服务器响应的 location 中变量$host 和 $http_host 便发生了变化。

指令名称：more_clear_input_headers

语法：more_clear_input_headers [-t <content-type list>]... <header>...

默认值：no

使用环境：http，server，location，location 中 if 区段

功能：清除指定的输入头。

> 实际上，more_clear_input_headers -t'text/plain'FooBaz 完全等同于 more_set_input_headers -t 'text/plain' "Foo: " "Baz: "或者 more_set_input_headers -t 'text/plain' Foo Baz。

从 v0.10 版本起，ngx_headers_more 模块的 more_clear_input_headers 和 more_clear_headers 指令能够将指定的头彻底删除。在早期的版本中，清除头通常指的是仅仅清除头的值，而不是一同清除，例如：

```
more_clear_headers 'Server';
```

最后的结果是：

```
Server:
```

就是说这个“Server：”仍然在响应头中，而从 v0.10 版起，已经被全部清除了。一些非标准的头，像 X-Foo，同样也能够将其完全清除。

对于 more_clear_input_headers 指令，同样是这样的，能够将进入的请求中的某些头彻底清除后，然后再将其转发到内容请求程序，例如，“proxy_pass”或者“fastcgi_pass”，等等。

4. 使用实例

在 Nginx 的配置文件中添加如下配置内容：

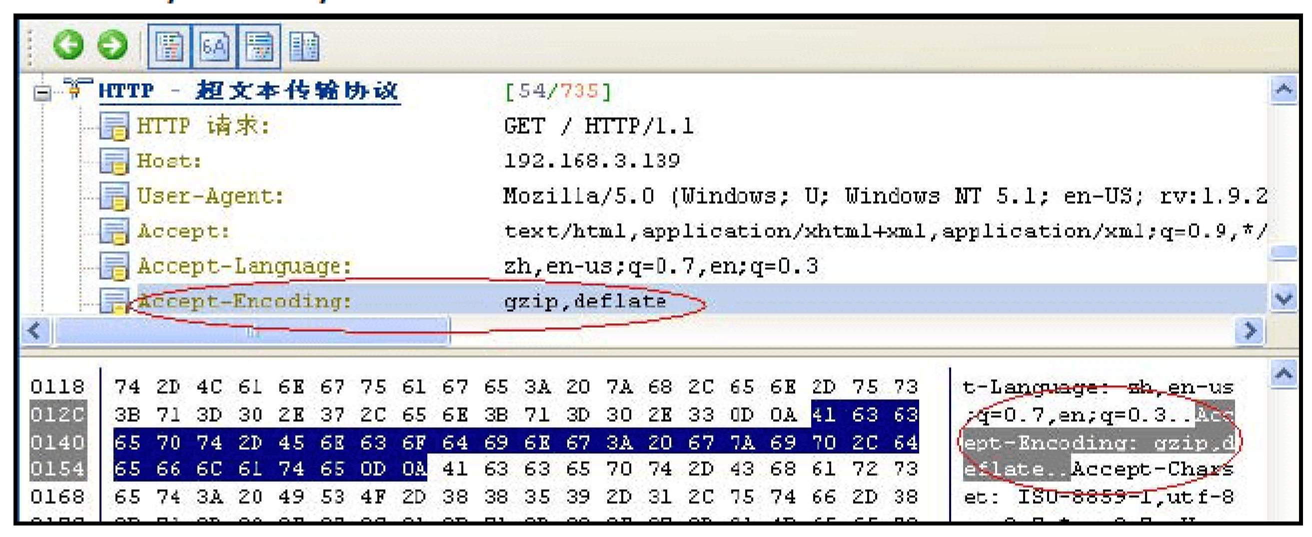
```
location / {
more clear input headers 'transfer-encoding';
proxy pass http://192.168.3.246;
}
```

在将其发到后台服务器之前将 transfer-encoding 清除。

5. 相关的 header

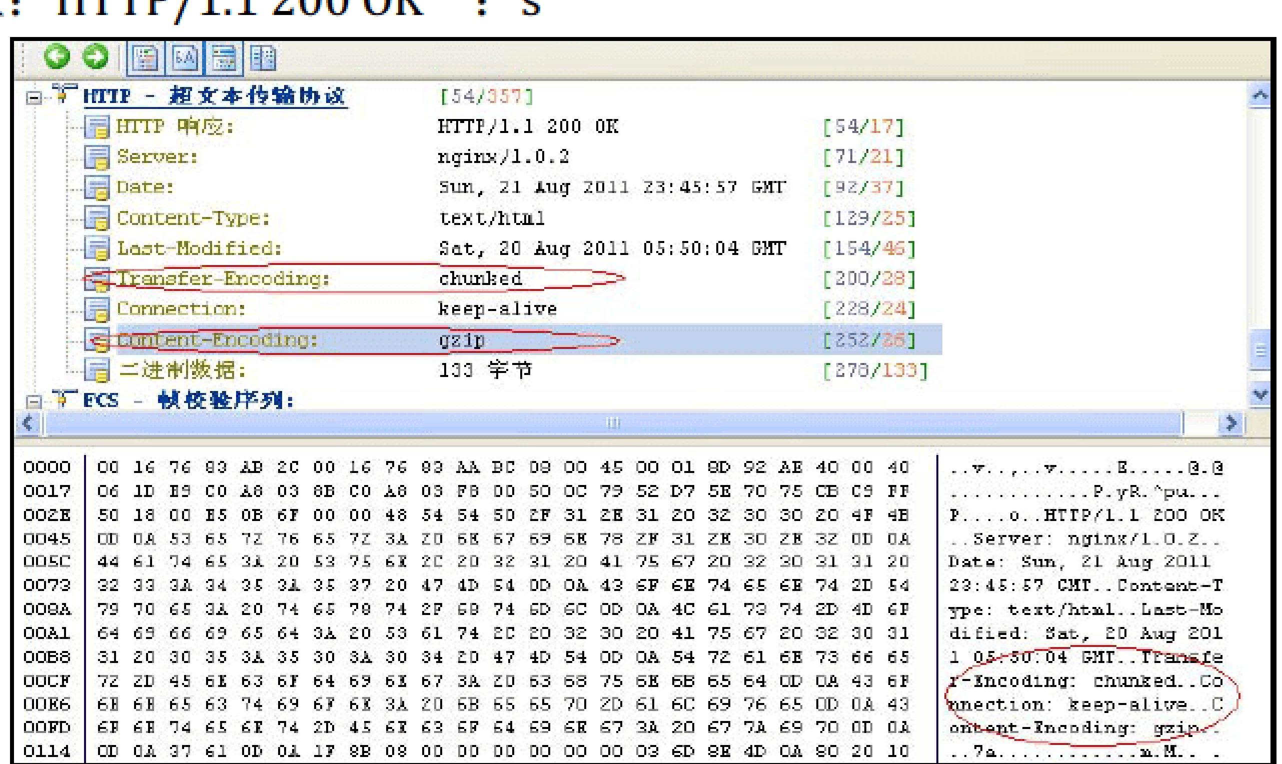
客户端发出的请求

“HTTP 请求：GET / HTTP/1.1”：



服务器返回的响应

“HTTP 响应：HTTP/1.1 200 OK”：s

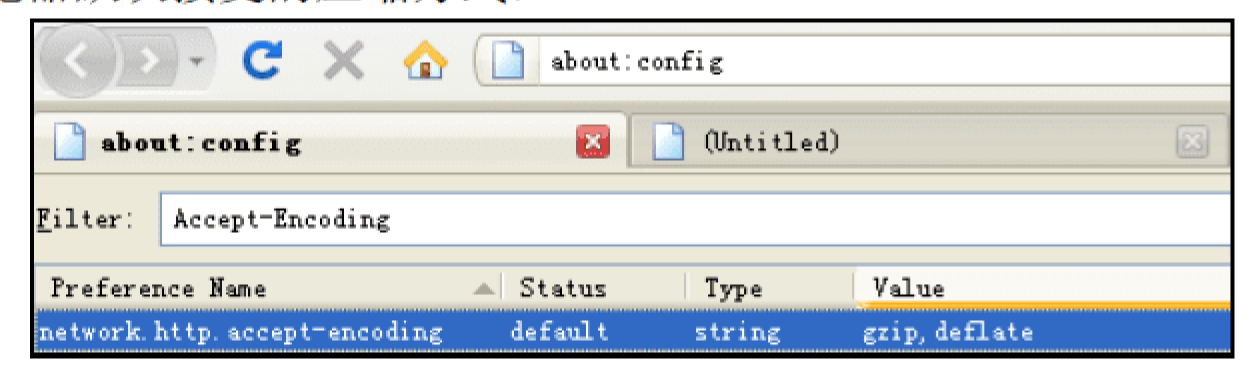


在上面两个截图中三个圈起的地方，我们看到了三种类型的编码：Accept-Encoding、Content-Encoding 和 Transfer-Encoding，下面我们看一下这三种编码的作用。

Accept-Encoding

这是客户端即用户代理（User-Agent，一般就是我们使用的浏览器）发出请求头告诉服务器端，客户端可支持的压缩方式。需要注意的是，如果客户端没有指定具体的压缩方式，就是说在客户端的请求中没有 Accept-Encoding 头，并不代表客户端不接受压缩方式，而是说，服务器端可以使用任何压缩方式，客户端都可以接受任何编码类型，压缩或者非压缩，即压缩中的任何类型。

这是 firefox 浏览器默认接受的压缩方式：



Content-Encoding

在服务器返回的响应中，我们注意到“Content-Encoding: gzip”，这表示在服务器向客户端发送的响应中已经使用的编码方式。对于 Content-Encoding 的值，在 HTTP1.1 的标准下，有 gzip、compress、deflate、identity 这些取值，这些取值对大小写不敏感。服务器端的 Content-Encoding 和客户端的 Accept-Encoding 是相对应的，客户端的请求中指定了“Accept-Encoding: gzip, deflate”，而 Nginx 服务器端的配置文件中设置了“gzip on;”，而且指定了压缩的 MIME 类型，例如“gzip_types text/plain application/xml;”，那么在传输相应类型的文件中将会使用 gzip 传输。

Transfer-Encoding

当服务器生成的 HTTP 响应无法在响应头中确定要传输数据的大小时，服务器端便不会提供 Content-Length 的头信息，而是通过 Chunked 编码的方法来提供。在前面我们也见过“Transfer-Encoding: chunked”这种格式，如果在响应头中出现了这种头，那么我们要传输的内容部分将会是以 chunked 的方式进行传输。有 chunked 编码传输，就需要 un-chunked 来解码。

在 Nginx 中，默认开启了“Transfer-Encoding: chunked”，它会根据发送的需要自动使用，对于使用这种方式是如何切分数据，以及确定数据传输完成，其实也很简单。假如我们对一根 6.5 厘米的黄瓜进行切割，每一段 1 厘米，那么切到第八刀的时候就没有了，第七刀就是不够标准大小，但是也是有数据，但是第八刀的时候数据就为 0 了，那么这就是结束标志。

Content-type

内容类型（Content-Type），这个头部领域用于指定消息的类型，格式为：

```
Content-Type: [type]/[subtype]; parameter
```

在前面介绍的四条指令中 <content-type list>，这里的 list 可取的值如下：

类型/之类型	文件的扩展名
text/html	html htm shtml;
text/css	css;
text/xml	xml;
image/gif	gif;
image/jpeg	jpeg jpg;
application/x-javascript	js;
application/atom+xml	atom;
application/rss+xml	rss;
text/mathml	mml;
text/plain	txt;
text/vnd.sun.j2me.app-descriptor	jad;

续表

类型/之类型	文件的扩展名
text/vnd.wap.wml	wml;
text/x-component	htc;
image/png	png;
image/tiff	tif tiff;
image/vnd.wap.wbmp	wbmp;
image/x-icon	ico;
image/x-jng	jng;
image/x-ms-bmp	bmp;
image/svg+xml	svg;
application/java-archive	jar war ear;
application/mac-binhex40	hqx;
application/msword	doc;
application/pdf	pdf;
application/postscript	ps eps ai;
application/rtf	rtf;
application/vnd.ms-excel	xls;
application/vnd.ms-powerpoint	ppt;
application/vnd.wap.wmlc	wmlc;
application/vnd.wap.xhtml+xml	xhtml;
application/vnd.google-earth.kml+xml	kml;
application/vnd.google-earth.kmz	kmz;
application/x-7z-compressed	7z;
application/x-cocoa	cco;
application/x-java-archive-diff	jardiff;
application/x-java-jnlp-file	jnlp;
application/x-makeself	run;
application/x-perl	pl pm;
application/x-pilot	prc pdb;
application/x-rar-compressed	rar;
application/x-redhat-package-manager	rpm;
application/x-sea	sea;
application/x-shockwave-flash	swf;
application/x-stuffit	sit;
application/x-tcl	tcl tk;
application/x-x509-ca-cert	der pem crt;
application/x-xpinstall	xpi;
application/zip	zip;
application/octet-stream	bin exe dll;
application/octet-stream	deb;
application/octet-stream	dmg;
application/octet-stream	eot;

续表

类型/之类型	文件的扩展名
application/octet-stream	iso img;
application/octet-stream	msi msp msm;
audio/midi	mid midi kar;
audio/mpeg	mp3;
audio/x-realaudio	ra;
video/3gpp	3gpp 3gp;
video/mpeg	mpeg mpg;
video/quicktime	mov;
video/x-flv	flv;
video/x-mng	mng;
video/x-ms-asf	asx asf;
video/x-ms-wmv	wmv;
video/x-msvideo	avi;

第 23 章　重写 URI

我们在使用 Nginx 的过程中经常碰到需要将客户端访问者的 URI 进行重写，在“根据浏览器选择主页”部分对这种做法有了初步的认识，另外我们在“map 模块的使用”部分也见到和改写过，但是它们都没有 rewrite 模块强大，因此，在这一部分我们看看 rewrite 的用法。

使用 rewrite 模块离不开正则表达式，因此，要想使用 rewrite 指令，就必须在安装 Nginx 时指定 Pcre。

使用 rewrite 模块通过正则表达式（Pcre）就可以改变 URI，并且可以重定向和根据变量来选择配置。如果在 server 级别执行 rewrite 指令，那么请求将在 location 确定之前执行。如果在被选择的 location 中仍有 rewrite 指令，那么它们同样被执行，如果在这个 location 中又触发访问到 rewrite 指令，那么就会再次改变 URI。这种被重复周期为 10 次， 在 10 次之后如果仍然找不到具体的 URI，那么 Nginx 将会返回 500 错误。

1. 配置示例

```
  rewrite ^/ (/d+) / (.+) / /$2?id=$1 last;

  rewrite ^/ ([0-9a-z]+) job/ (.*) $ /area/$1/$2 last;

  if  (-d $request_filename) {
rewrite ^/ (.*)  ([^/]) $ http://$host/$1$2/ permanent;
  }

  if  ($http host ~* "^ (.*) /.st/.xx/.com$")  {
  rewrite ^ (.*)  http://st.yy.com$1;
  break;
  }

  if  ($host ~*  (.*) \.xx\.com)  {
   set $sub_name $1;
   rewrite ^/sort\/ (\d+) \/?$ /index.php?act=sort&cid=$sub name&id=$1 last;
   }

  if  ($http_user_agent ~ MSIE)  {
   rewrite ^ (.*) $ /var/www/ie/$1 break;
  }
```

2. 安装 pcre

下载并安装 pcre：

```
[root@s29 ~]# wget ftp://ftp.csx.cam.ac.uk/pub/software \
> /programming/pcre/pcre-8.13.tar.gz
```

```
[root@s29 ~]#tar -zxvf pcre-8.13.tar.gz
[root@s29 ~]#cd pcre-8.13
[root@s29 pcre-8.13]#./configure --prefix=/usr/local/pcre-8.13   \
> --enable-utf8 --enable-unicode-properties
[root@s29 pcre-8.13]#make
[root@s29 pcre-8.13]#make install
```

需要注意的是，在安装 Nginx 时：

```
[root@mail nginx-1.0.2]# ./configure --prefix=/usr/local/nginx-1.0.2
-pcre-8.13 \
> --with-pcre=/root/pcre-8.13

  ...

Configuration summary
  + using PCRE library: /root/pcre-8.13
  + OpenSSL library is not used
  + md5: using system crypto library
  + sha1 library is not used
  + using system zlib library

  ...
```

这里的--with-pcre 指定的是 pcre-8.13 的源码，但是在 Nginx 的安装过程中会查找 pcre 安装的具体情况：

```
[root@mail nginx-1.0.2]#make

  ...

pcre-8.13 configuration summary:

Install prefix .................. : /usr/local
C preprocessor .................. : gcc -E
C compiler ...................... : gcc
C++ preprocessor ................ : g++ -E
C++ compiler .................... : g++
Linker .......................... : /usr/bin/ld
C preprocessor flags ............ :
C compiler flags ................ : -O2 -fomit-frame-pointer -pipe
C++ compiler flags .............. : -O2
Linker flags .................... :
Extra libraries ................. :

Build C++ library ............... : yes
```

```
Enable UTF-8 support ........... : yes
Unicode properties ............. : no
Newline char/sequence .......... : lf
\R matches only ANYCRLF ........ : no
EBCDIC coding .................. : no
Rebuild char tables ............ : no
Use stack recursion ............ : yes
POSIX mem threshold ............ : 10
Internal link size ............. : 2
Match limit .................... : 10000000
Match limit recursion .......... : MATCH_LIMIT
Build shared libs .............. : no
Build static libs .............. : yes
Buffer size for pcregrep ....... : 20480
Link pcregrep with libz ........ : no
Link pcregrep with libbz2 ...... : no
Link pcretest with libreadline .. : no

  ...
```

3. 指令

rewrite 模块提供了 7 条指令，以便更好地控制 URI 重定向。

指令名称：break

语法：break

默认值：none

使用环境：server，location，if

功能：完成当前的规则组，不再处理任何其他 rewrite 指令。例如：

```
if  ($slow)  {
  limit rate  10k;
  break;
}
```

指令名称：if

语法：if （condition）{ ... }

默认值：none

使用环境：server，location

功能：用于检测条件是否成立，如果条件被评估为真，那么在大括号“{}”中代码将会被执行，同配置中一致的请求将会被处理。if 指令内部的配置会被从上一级继承。

> 需要注意的是，使用 if 指令要谨慎，尽量考虑使用 try_files 指定。

条件 condition 部分可以指定下列值：

- 使用= 或 !=操作符比较变量的表达式。

变量名称，如果变量名称为 false，那么值将会是一个空字符串（“”），或者是任何以“0”开始的字符串使用正则表达式的模式匹配：

  - ~：执行区分大小写匹配；
  - ~*：执行不区分大小写匹配；
  - !~：执行区分大小写不匹配；
  - !~*：执行不区分大小写不匹配。

- 使用 -f 或者 !-f 操作符检测文件的存在性。
- 使用 -d 或者 !-d 操作符检测目录的存在性。
- 使用 -e 或者 !-e 操作符检测文件、目录或者符号连接的存在性。
- 使用 -x 或者 !-x 检测文件是否可执行。

正则表达式部分可以放在括号内，以便在以后的使用中可以通过变量$1 ～ $9 来访问。例如：

```
if ($http_user_agent ~ MSIE) {
  rewrite  ^(.*)$  /msie/$1  break;
}

if ($http_cookie ~* "id=([^;] +) (?:;|$)" ) {
  set  $id  $1;
}

if ($request_method = POST ) {
  return 405;
}

if ($slow) {
  limit_rate  10k;
}

if ($invalid_referer) {
  return   403;
}

if ($args ~ post=140){
  rewrite ^ http://example.com/ permanent;
}
```

在条件判断中可以使用的一些全局变量：$args、$content_length、$content_type、$document_root、$document_uri、$host、$http_user_agent、$http_cookie、$limit_rate、$request_body_file、$request_method、$remote_addr、$remote_port、$remote_user、$request_filename、$request_uri、$query_string、$scheme、$server_protocol、$server_addr、$server_name、

$server_port、$uri。

指令名称：return

语法：return code

默认值：none

使用环境：server，location，if

功能：该指令会结束执行规则，并且会为客户端返回状态码。可使用的代码值有：204、400、402～406、408、410、411、413、416 和 500～504。此外，非标准的代码 444 将会关闭连接而不会发送任何头。

指令名称：rewrite

语法：rewrite regex replacement flag

默认值：none

使用环境：server，location，if

功能：该指令会按照相关的 regex 正则表达式和 replacement 替换字符串改变 URI。rewrite 指令会按照自己在配置文件中出现的顺序执行，这一点要格外注意。

如果 replacement 替代字符串由 http://开始，那么客户端将会被重定向 redirect，任何其他后面的 rewrite 指令都被终结。

标志 flag 用于结束 rewrite 指令，它的可取值如下。

- last：在搜索到相应的 URI 和 location 之后完成 rewrite 指令；
- break ：完成 rewrite 指令处理；
- redirect ：返回 302 临时重定向，如果 replacement 替换部分是由 http://开始，它将被应用；
- permanent：返回 30 代码永久重定向。

注意，如果重定向是相当的，即没有主机部分，那么当使用重定向时，Nginx 使用“Host”头的顺序为：如果有匹配 server_name 指令指定的主机名，那么则使用它；如果没有，那么将会使用第一个 server_name 设置的值；如果仍然没有，那么本地主机名会被使用。如果想让 Nginx 总是使用“Host”头，那么可以在 server_name 指令中使用“*”（但是这么做会有限制）。例如：

```
rewrite ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  last;
rewrite ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   last;
return  403;
```

如果我们将这些指令放置在/download/中，那么有必要将标志“last”替换为“break”，否则 Nginx 在经过 10 次循环后将会返回 500 错误：

```
location /download/ {
  rewrite ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
  rewrite ^(/download/.*)/audio/(.*)\..*$  $1/mp3/$2.ra   break;
  return  403;
}
```

如果在 replacement 替换部分包含参数，那么其余的参数添加在后面。另外，为了避免参数部分再被附加，可以在参数部分最后一个字符后再放置一个“？”。例如：

```
rewrite  ^/users/(.*)$  /show?user=$1?  last;
```

注意使用大括号（{和}），由于它们可以同时使用在正则表达式和区段（就是 http、server、location、if）控制中，为了避免歧义、冲突，因此在正则表达式中使用大括号（{和}）并将它们使用双引号或者单引号括起来。例如：

我们将/photos/123456 重定向该 URL 到/path/to/photos/12/1234/123456.png，那么可以使用下面的指令（注意在正则表达式中使用了双引号）：

```
rewrite  "/photos/([0-9]{2})([0-9]{2})([0-9]{2})"  /path/to/photos/$1
/$1$2/$1$2$3.png;
```

如果在 rewrite 结尾处使用了一个问号“？”，那么 Nginx 将会丢弃原有的$args 参数。当使用$request_uri 或者 $uri&$args 的时候，我们应该在 rewrite 结尾处使用“？”，以避免 Nginx 两次使用查询字符串。

例如，看下面的例子中，我们使用$request_uri，将 www.example.com 重定向为 example.com：

```
server {
  server name www.example.com;
  rewrite ^ http://example.com$request_uri? permanent;
}
```

指令 rewrite 同样只能对路径进行重定向，不会对参数操作。因此，要将一个有参数的 URL 重定向到另一个 URL，那么使用以下方式替代：

```
if ($args ^~ post=100){
  rewrite ^ http://example.com/new-address.html? permanent;
}
```

然而，需要注意的是，$args 变量不能被编码，不像在 location 中匹配的 URL 一样。

指令名称：rewrite_log

语法：rewrite_log on | off

默认值：rewrite_log off

使用环境：http，server，if-in-server，location，if-in-location

功能：如果启用该指令，那么将在错误日志中记录 notice 级别的 rewrite 日志。

指令名称：set

语法：set variable value

默认值：none

使用环境：server，location，if

功能：该指令用于为指定的变量创建一个值，它的值可以是文本、变量或者是它们的组合。可以使用该指令来定义一个新的变量，但是不能设置$http_xxx 头变量的值。

指令名称：uninitialized_variable_warn

语法：uninitialized_variable_warn on|off

默认值：uninitialized_variable_warn on

使用环境：http，server，location，if

功能：开启或关闭记录有关没有被初始化的变量。实际上，指令在配置文件载入内部代码的

时候就被包含在内，但是在请求使用时才会被解释器解释。下面我们看一个虚拟机的堆栈解释，配置如下：

```
location /download/ {
  if ($forbidden) {
return   403;
  }
  if ($slow) {
limit rate  10k;
  }
  rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
}
```

被编译的顺序如下：

```
变量 $forbidden
检查为 0
recovery 403
完成整条代码
变量 $slow
检查为 0
检查正则表达式
复制 "/"
复制$1
复制"/mp3/"
复制$2
复制".mp3"
完成正则表达式
完成整条序列
```

> 注意，在这个解释序列中没有出现 limit_rate，这是因为该指令没有涉及到 ngx_http_rewrite_module。如果在配置文件中同样的部分存在“if”区段，那么它将会被作为“location”指令。

如果变量$slow 为 true，那么“if”区段的指令将会被执行，当然具体到这个配置中，那么就是将 limit_rate 的值设置为 10k。

```
rewrite  ^/(download/.*)/media/(.*)\..*$  /$1/mp3/$2.mp3  break;
```

如果我们将正则表达式的第一个斜线“/”包含在括号内，那么将会减少指令序列：

```
rewrite  ^(/download/.*)/media/(.*)\..*$  $1/mp3/$2.mp3  break;
```

再看一下现在的指令执行序列：

```
检查正则表达式
复制$1
复制"/mp3/"
复制$2
复制".mp3"
```

```
    完成正则表达式
    完成整个代码序列
```

4. 使用实例

这是 Nginx 配置文件中的一部分：

```
server {
  listen   80;
  server_name  www.xx.com;
  # ssi on ;
  index index.shtml index.htm index.php;
  include   /cf/g.conf


  location /download {
root   /sdc/www/news/;
index  index.html index.htm index.php;
  }

  if  ( $host ~ zt.xx.com)  {
 rewrite  /$   /zt/dfh/;
 rewrite  /index.shtml$   /zt/dfh/index.shtml;
 }

  if  ( $host ~ df.xx.com )  {
 rewrite  /$   /index.shtml;
 rewrite  /index.shtml   /df/bj/index.shtml;
 rewrite  ^(.+)shtml$   http://news.xx.com/news/$1shtml;
 }

  if  ( $host ~ www.xx.com.cn )  {
 rewrite  /$  /index.shtml;
 rewrite  ^(.+)shtml$ http://www.xx.com$1shtml;
 rewrite  ^(.+)html$  http://www.xx.com$1html;
 rewrite  ^(.+)htm$   http://www.xx.com$1htm;
 }

  if  ( $host ~ www.bjtimes.cn )  {
 rewrite  /$   /index.shtml;
 rewrite  ^(.+)shtml$ http://www.jinghua.cn$1shtml;
 rewrite  ^(.+)html$  http://www.jinghua.cn$1html;
 rewrite  ^(.+)htm$   http://www.jinghua.cn$1htm;
 }
```

```
 if ( $host ~ yy.com) {
rewrite /$  /index.shtml;
rewrite ^(.+) shtml$ http://www.yy.com$1shtml;
rewrite ^(.+) html$  http://www.yy.com$1html;
rewrite ^(.+) htm$   http://www.yy.com$1htm;
}

 if ( $host ~ xx.com.cn ) {
rewrite ^(.+) shtml$  http://www.xx.com/;
rewrite ^(.+) html$   http://www.xx.com/;
}

  if ( $host ~ rw.xx.com) {
rewrite /$  /index.shtml;
rewrite /index.shtml  http://news.xx.com/index.shtml ;
rewrite ^(.+) shtml$http://news.xx.com/rw/$1shtml ;
}

  if ( $host ~ g.xx.com ) {
rewrite /$  /index.shtml;
rewrite ^(.+) shtml$  /g/$1shtml ;
}

  if ( $host ~ t.xx.com ) {
rewrite /$  /index.shtml;
rewrite ^(.+) shtml$  /t/$1shtml ;
}


location /php
  {
access_log  off;
ssi off ;
proxy pass http://192.168.10.15/p;
proxy set header  Host $host;
proxy set header  X-Real-IP$remote addr;
proxy set header  X-IP$remote addr;
proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;
  }

 location /pic/
  {
ssi off ;
```

```
    proxy_pass  http://192.168.10.16/;
       }


      error page  404 = http://www.jinghua.cn/404.shtml
    }

    server{
      listen   80;
      server_name  r.xx.com
      indexindex.shtml index.htm index.php;
      include   /cf/g.conf

      rewrite  ^/r.php$   /php/rss/r.php ;
      rewrite  ^/n.php$   /php/rss/rss.xml ;
      rewrite  ^/h.php$   /php/rss/h.php ;
      rewrite  ^/index_rollsnews.shtml$ /php/rss/news.php;

      rewrite  ^ (.+) shtml$   /t/$1shtml ;
    }
```

邪恶的 if

当将 if 指令使用在 location 区段时可能会有问题，在有些情况下，它不会按照我们的期望去做，有些情况下却是完全不同，有些情况下甚至会产生故障。通常的做法就是尽可能地避免使用 if。

在 location 区段中：

```
 * return …;
 * rewrite … last;
```

任何其他的情况都可能引起不可预测的行为，包括潜在的 SIGSEGV（SIGSEGV 是当一个进程执行了一个无效的内存引用，或发生段地址错误时发送给它的信号）。

> 特别需要注意的是，对于给定的两个同一请求，将不会随机地失败一个而另一个则工作，就是说 if 的行为不会一致。因此，要适当地测试和理解 if 后才可以使用它。

下面的例子中，是一个简单的却又不能避免使用 if 的例子：

```
    if  ($request_method = POST )  {
      return 405;
    }
    if  ($args ~ post=140) {
      rewrite ^ http://example.com/ permanent;
    }
```

在这个例子，如果想测试一个变量，却没有等效的指令。

替代 if

如果 try_files 适合我们的需要，那么就用 try_files，在其他情况下使用“return …”或者“rewrite … last”。在有些情况下，也可能会将 if 移动到 server 级别的区段，在 server 级别是安全的，而且 rewrite 模块的某些指令也只能到这一级别，就是说在高的级别，例如 http 级别就不能使用了。

例如，下列配置可能被用于安全地改变 location，将用于请求的处理：

```
location / {
  error_page 418 = @other;
  recursive_error_pages on;

  if ($something) {
return 418;
  }

  # some configuration
  …
  }

location @other {
   # some other configuration
 …
}
```

在某些情况下，使用嵌入的脚本模块（嵌入的 perl 或者是其他的第三方模块）通过脚步来完成。

出问题的例子

下面是一些例子，解释为什么 if 不太好用，不要尝试使用，会被警告：

```
# Here is collection of unexpectedly buggy configurations to show that
# if inside location is evil.
在这里收集的都是配置 if 时不期望出现的情况，说明了 if 的拙劣行为


# 在响应中仅第二个头将会被添加，这种情况不属于真正的 bug，只是说明它是如何工作

location /only-one-if {
  set $true 1;

  if ($true) {
 add_header X-First 1;
  }

  if ($true) {
 add_header X-Second 2;
```

```
    }

    return 204;
    }

  #由于 if，请求的 uri 没有改变为 '/' 就被发送到后台。

  location /proxy-pass-uri {
    proxy_pass http://127.0.0.1:8080/;

    set $true 1;

    if ($true) {
   # nothing
    }
  }

  # 由于 if、try_files 将不会工作

  location /if-try-files {
    try_files  /file  @fallback;

    set $true 1;

    if ($true) {
  # nothing
    }
   }

  # Nginx 将会产生 SIGSEGV

  location /crash {
     set $true 1;

     if ($true) {
     # fastcgi_pass here
     fastcgi_pass  127.0.0.1:9000;
     }

     if ($true) {
     # no handler here
     }
  }
```

```
# 通过捕获的别名不会正确地继承到由 if 创建的隐式嵌套的 location 中

location ~* ^/if-and-alias/ (?<file>.*) {
  alias /tmp/$file;

  set $true 1;

  if ($true) {
  # nothing
}
 }
```

期待修复

指令“if”是 rewrite 模块的一部分，用于评估指令执行，另一方面，一般而言 Nginx 配置文件是声明的，但是在某些时候，由于用户需求尝试，就会在“if”中使用一些非 rewrite 指令，这就导致了我们现在所讨论的情况。几乎都会正常的工作，但是我们也看到了上面的一些不尽如人意的例子，看起来似乎只有正确地修复这个缺陷，在 if 内部完全禁用非 rewrite 指令，这将会对现有的程序的构造造成很大的破坏，因此还没有完成。

结论

如果你读完以上的内容，但是仍然想使用 if，那么要注意以下两点：

- 确保理解 if 是如何工作的；
- 一定要进行适当的测试。

第 24 章　Nginx 与服务器端包含

Nginx 服务器通过 HttpSsiModule 模块来提供服务器端包含处理。该模块提供了一个过滤器，用于处理服务器端包含（SSI），它当前支持的 ssi 命令还并不完善。

1. 配置示例

```
location / {
  ssi  on;
}
```

2. 指令

HttpSsiModule 模块提供了 4 条指令。

指令名称：ssi

语法：ssi [on | off]

默认值：ssi off

使用环境：http，server，location，if in location

功能：启用 SSI 处理。

> 注意，当 ssi 开启时，将不再发送 Last-Modified 和 Content-Length 头。

例如，在配置文件中添加以下配置：

```
 location ~* \.shtml$ {
ssi on;
 }
```

指令名称：ssi_silent_errors

语法：ssi_silent_errors [on|off]

默认值：ssi_silent_errors off

使用环境：http，server，location

功能：如果开启该指令，即设置为 on，如果在处理 ssi 指令时发生错误，那么将不会输出“[an error occurred while processing the directive”错误。

指令名称：ssi_types

语法：ssi_types mime-type [mime-type ...]

默认值：ssi_types text/html

使用环境：http，server，location

功能：除了“text/html”类型外，对其他的 MIME 类型开启 SSI 处理功能。

指令名称：ssi_value_length

语法：ssi_value_length length

默认值：ssi_value_length 256

使用环境：http，server，location

功能：该指令用于定义在 SSI 中允许参数值的长度。

24.1 ssi 指令

我们首先需要认识 ssi 指令的使用格式：

```
<!--# command parameter1=value parameter2=value ... parameterN=value -->
```

> 注意，井号（#）必须紧跟在双破折号（注意是英语的符号，其实我们叫双短线更合适）之后。下面我们了解一下 Nginx 支持的 ssi 指令。

1. 指令

Nginx 的 SSI 模块可以使用以下 6 条 ssi 指令。

指令名称：block

功能：该指令可以用来创建一个块（或者也可以叫做区段），这个块可以被用于 include 指令。在块内还可以使用其他的 ssi 指令。该指令有一个参数 name，它用来表示定义块的名称。

例如：

```
<!--# block name="one" -->
  the silencer
<!--# endblock -->
```

指令名称：config

功能：该指令用于为 SSI 指定一些配置参数。

- errmsg：用于指定的内容将会在 SSI 处理过程中出现错误时输出。默认字符串为：[an error occurred while processing the directive]。
- timefmt：用于设置时间格式，如果想在时间中包含秒，那么也可以使用“%s”格式。默认的格式为：

```
"%A, %d-%b-%Y %H:%M:%S %Z"
```

语法：

```
<!--#config errmsg="自定义错误信息"-->
<!--#config timefmt="显示时间格式"-->
```

例如：

```
  <!--# config errmsg="Sorry! Server SSI wrong, please contact me, Thanks"
-->
```

```
<!--# config  timefmt="%Y-%b-%d %H:%M:%S, %A %Z" -->
```

指令名称：echo

功能：显示一个变量。

- var：变量的名字。

- default：如果变量为空，那么显示这个字符串。默认为“none”。

例如：

```
<!--# echo var="name" default="no" -->
```

等同于：

```
<!--# if expr="$name" --><!--# echo var="name" --><!--# else -->no<!--#
endif -->
```

指令名称：if / elif / else / end

功能：根据条件包含文本或者其他指令。

语法：只能嵌套一层。

```
<!--# if expr="…" -->
  ...
<!--# elif expr="…" -->
...
<!--# else -->
...
<!--# endif -->
```

- expr：用于判定表达式，可以使用变量。如果在文本中使用了变量，那么它们将会被其值所取代。

使用变量：

```
<!--# if expr="$name" -->
```

字符串比较：

```
<!--# if expr="$name = text" -->
<!--# if expr="$name != text" -->
```

正则表达式：

```
<!--# if expr="$name = /text/" -->
<!--# if expr="$name != /text/" -->
```

例如：

```
<!--#if expr="$SERVER_NAME=\"mail.xx.com\""-->
  Hello, Welcome to mail.xx.com!
<!--#elif expr="$SERVER NAME=\"www.yy.com\"" -->
   Hello, Welcome to www.xx.com!
<!--#else-->
 Hello, Welcome to www.xx.com!
<!--#endif"-->
```

指令名称：include

功能：该指令用于包含另一个来源的内容，可选的来源有 file 或者 virtual。file 表示来源于文件，而 virtual 则表示来源于一个请求。

例如：

```
<!--# include file="footer.html" -->
```

```
<!--# include virtual="/remote/body.php?argument=value" -->
```

> 需要注意的是，file 或 virtual 的目标文件必须在 server 中配置的目录位置中，对于 virtual，给出的是到服务器端某个文档的虚拟路径，file 则只能是相对路径。

file 和 virtual 之间的主要区别是，如果 virtual 带有 wait 选项，那么和 file 是相同的。在这一点，使用上基本等同于 Apache，但是现在它们基本上可以执行相同的操作，两者都可以处理 URI 和静态文件。在并发中会导致多请求，如果想继续使用它们，那么要添加使用 wait 选项。

virtual 的参数：

- stub：该参数用于调用有 block 指令指定的 block 名称，如果请求为空或者是返回一个错误，那么将会使用该参数指定的值，也就是相应的 block 区段。例如：

```
<!--# block name="one" --> <!--# endblock -->
<!--# include virtual="/remote/body.php?argument=value" stub="one" -->
```

- wait ：如果将该变量设置为 yes，那么剩余的 SSI 将不会被判断，直到当前的请求完成。例如：

```
<!--# include virtual="/remote/body.php?argument=value" wait="yes" -->
```

指令名称：set

语法：<!--#set var="变量名" value="变量值"-->

- var：变量名称。
- value：变量值。

如果包含变量名，那么它们将进行判断。

功能：设置变量。

例如：

```
<!--#set var="color" value="RED"-->
```

2. 变量

可用的变量有两个。

- $date_local：表示本服务器的当前时间，可以使用 timefmt 指定日期的输出格式。
- $date_gmt：以格林尼治时间表当前的时间，可以使用 timefmt 指定日期的输出格式。

24.2 使用实例

在下面的两个例子中，我们了解 Nginx 在 HttpSsiModule 模块下.shtml 文件的执行过程和从 Nginx 服务器到客户端的传输情况。

1. 例 1

在 Nginx 的配置文件中添加以下配置：

```
http {
```

```
include   mime.types;
default_type  application/octet-stream;

sendfileon;
keepalive timeout  65;

    server {
  server_name www.xx.com;
  listen   80;
  root html;

 location / {
   ssi  on;
   #gzip  on;
 }

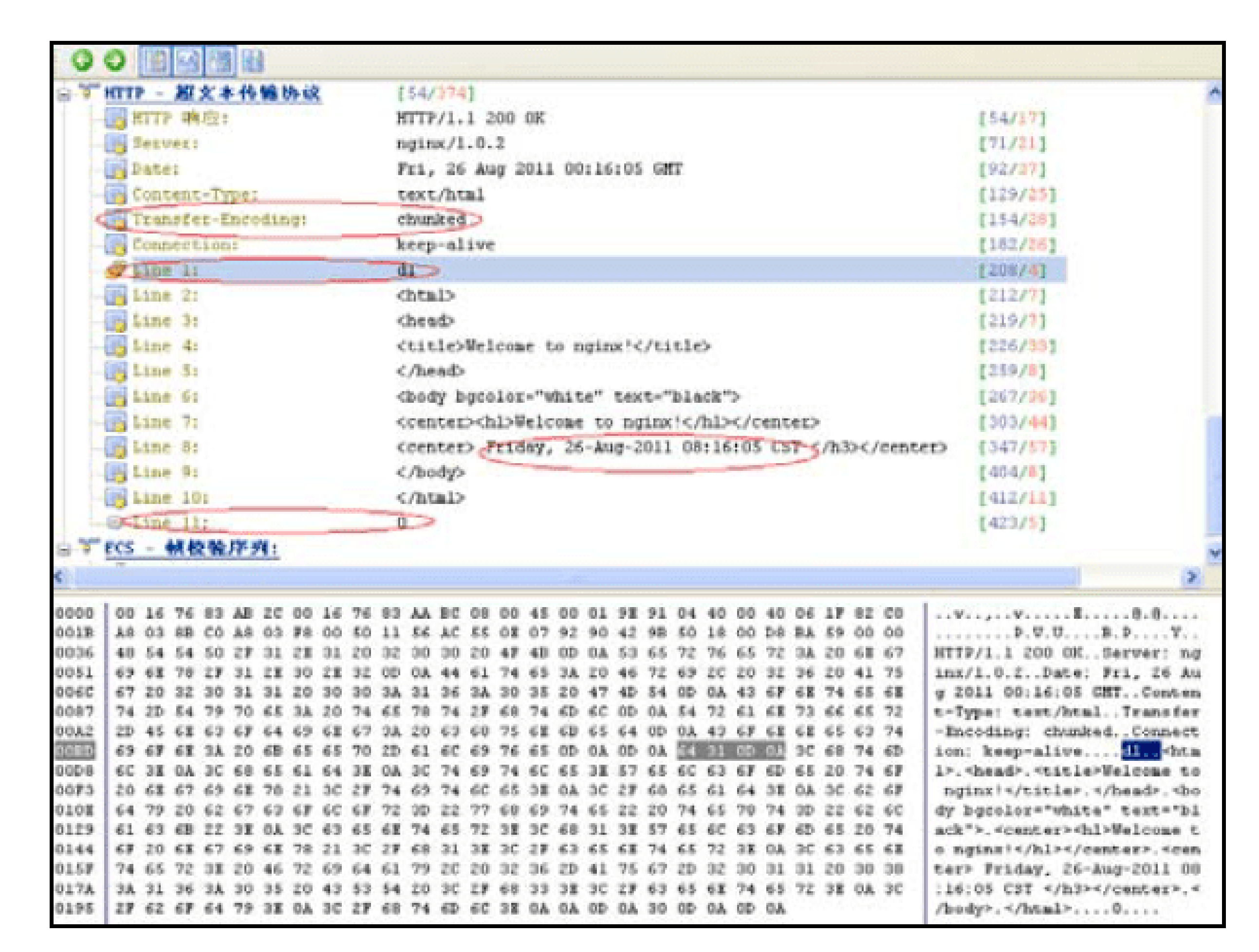
#location ~* \.shtml$ {
    #  ssi on;
    #}

    }
}
```

添加一个.shtml 文件：

```
[root@mail html]# cat index.shtml
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
<center> <!--#echo var="DATE_LOCAL"--> </h3></center>
</body>
</html>
```

我们看一下服务器端相应的截图：



从这个截图中，我们看出由于使用了 SSI，因此，在服务器端的响应中没有 Last-Modified 和 Content-Length 头。

2. 例 2

编写如下.shtml 文件：

```
[root@mail html]# vi  index.shtml

<!--# config  timefmt="%A" -->

<!--# if expr="$DATE_LOCAL=Monday" -->
<center><!--# include file="1.txt" --></center>

<!--# elif expr="$DATE_LOCAL=Tuesday" -->
<center><!--# include file="2.txt" --></center>


<!--# elif expr="$DATE_LOCAL=Wednesday" -->
<center><!--# include file="3.txt" --></center>

<!--# elif expr="$DATE_LOCAL=Thursday" -->
<center><!--# include file="4.txt" --></center>

<!--# elif expr="$DATE_LOCAL=Friday" -->
<center><!--# include file="5.txt" --></center>

<!--# else -->
  Hello, Welcome to www.xx.com!
<!--# endif -->

<html>
```

```
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
</body>
</html>

<!--# config errmsg="Sorry! Server SSI wrong, please contact me, Thanks" -->
<!--# config timefmt="%Y-%b-%d %H:%M:%S, %A %Z" -->

<center> <!--#echo var="DATE_LOCAL"--> </h3></center>
<center> <!--#echo var="date_gmt"--> </h3></center
```

在这个例子中我们从星期一到星期五分别调用不同的文件，显示不同的内容。该页面结尾的地方表现的是时间的使用方式，对于页面本身来说是没有什么用的，主要在于它们的使用方法。

下面通过两种方式访问，一种是 Telnet 方式，另一种是浏览器方式。

Telnet 方式

看一下访问结果（该输出结果完全没有改变，包括空格也没有删除）：

```
[root@jh-share ~]# telnet mail.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET /index.shtml HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Server: nginx/1.0.2
Date: Fri, 26 Aug 2011 08:45:31 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

f

<center>
2b
5555555555555555555555555555555555555555555

9c
</center>
```

```
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
<center>
4d
##########################################################################
#####

91
</center>
</body>
</html>


<center> 2011-Aug-26 16:45:31, Friday CST </h3></center>
<center> 2011-Aug-26 08:45:31, Friday GMT </h3></center>

0

Connection closed by foreign host.
```

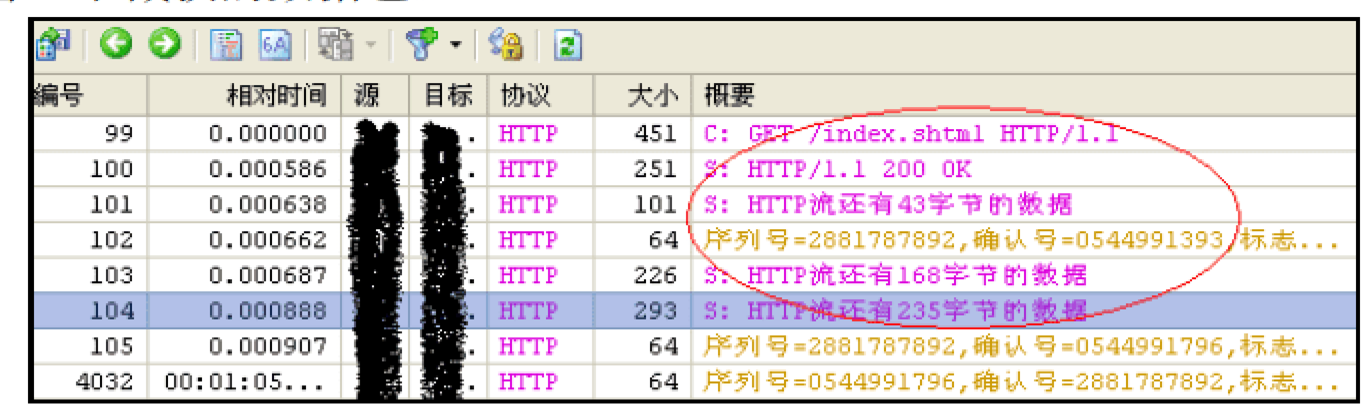
之所以分析这个例子就是要说明在 Nginx 解析该网页时的执行过程。对于.shtml 文件，其执行方式和 html 文件是一样的，都是从上到下的方式执行，但.shtml 文件与 html 文件不同的是，它不是统一得出页面的总长度，对于访问页面的最终结果而言，它是对每一个插入的对象分别计算。

例如，在上面的结果中，我们看一下 4d 处，这是一个长度值，以十六进制表示，换算为十进制就是 77 了。你可以数一下 4d 下面的“#”字符有多少个，如果你不想数，那么我告诉你，是 76 个，不信你就去数。那么另一个字符呢？那就是每一行最后的一个字符，叫它回车符、换行符都行，也就是不能显示的那个字符。

在页面结束的地方有一个数字 0，它表示页面内容输出结束。

浏览器方式

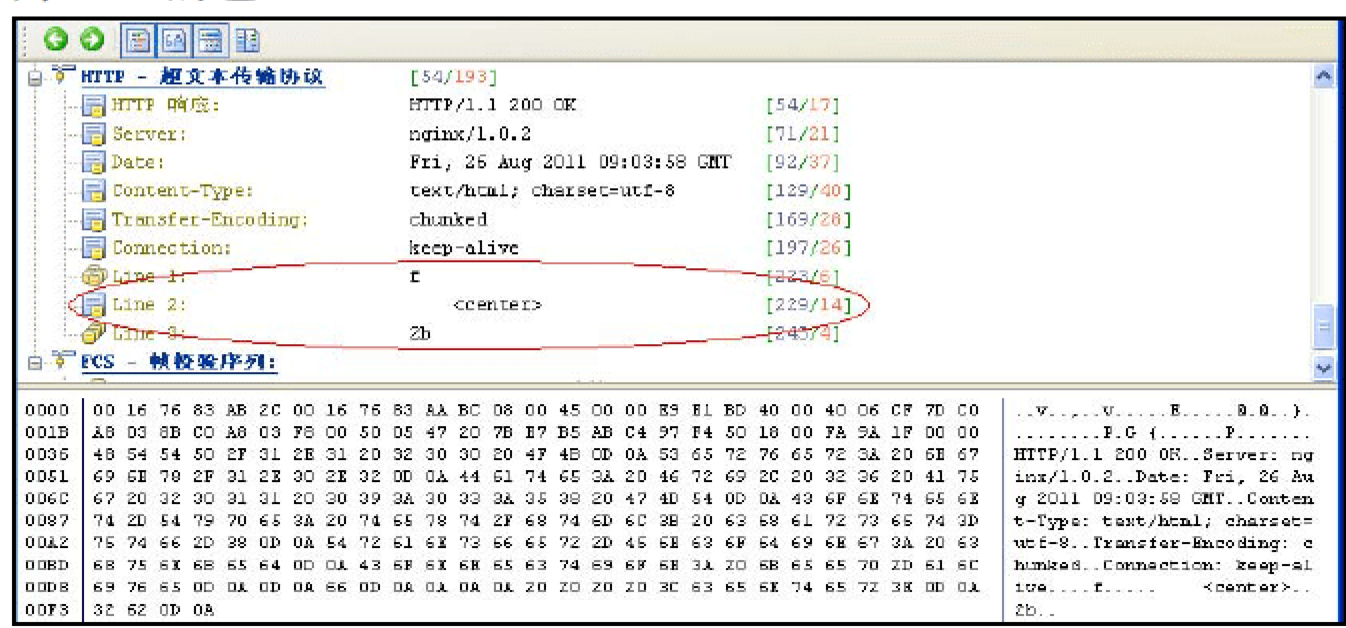
通过上面 Telnet 的方式，我们已经了解了 Nginx 对.shtml 文件的解释过程。但它是如何传输的呢？我们来看一下截获的数据包：



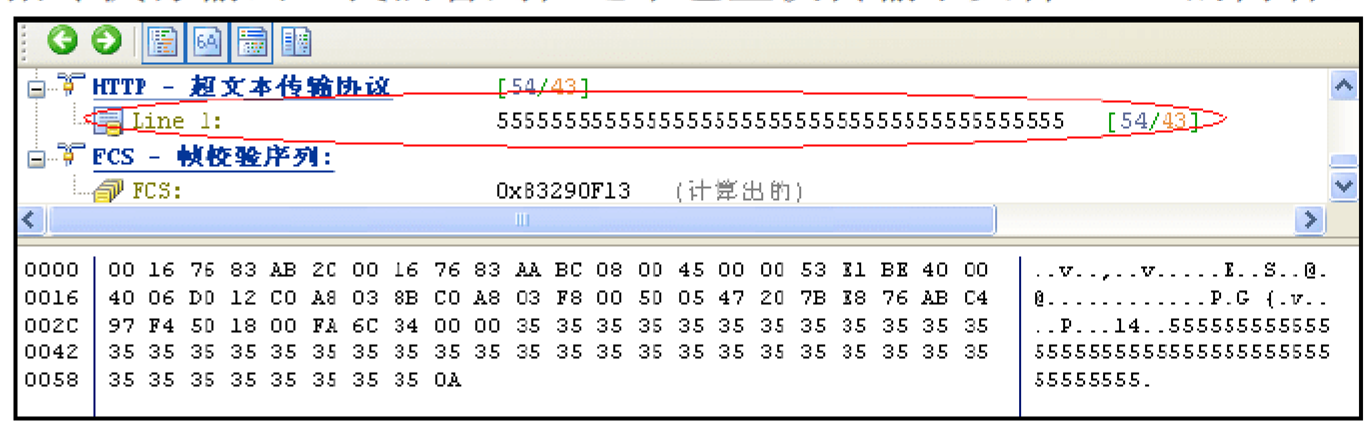
编号	相对时间	源	目标	协议	大小	概要
99	0.000000	[redacted]	[redacted]	HTTP	451	C: GET /index.shtml HTTP/1.1
100	0.000586	[redacted]	[redacted]	HTTP	251	S: HTTP/1.1 200 OK
101	0.000638	[redacted]	[redacted]	HTTP	101	S: HTTP流还有43字节的数据
102	0.000662	[redacted]	[redacted]	HTTP	64	序列号=2881787892,确认号=0544991393,标志...
103	0.000687	[redacted]	[redacted]	HTTP	226	S: HTTP流还有168字节的数据
104	0.000888	[redacted]	[redacted]	HTTP	293	S: HTTP流还有235字节的数据
105	0.000907	[redacted]	[redacted]	HTTP	64	序列号=2881787892,确认号=0544991796,标志...
4032	00:01:05...	[redacted]	[redacted]	HTTP	64	序列号=0544991796,确认号=2881787892,标志...

看编号部分，99 为客户端请求的数据包，100、101、103 和 104 为服务器返回的响应数据包。

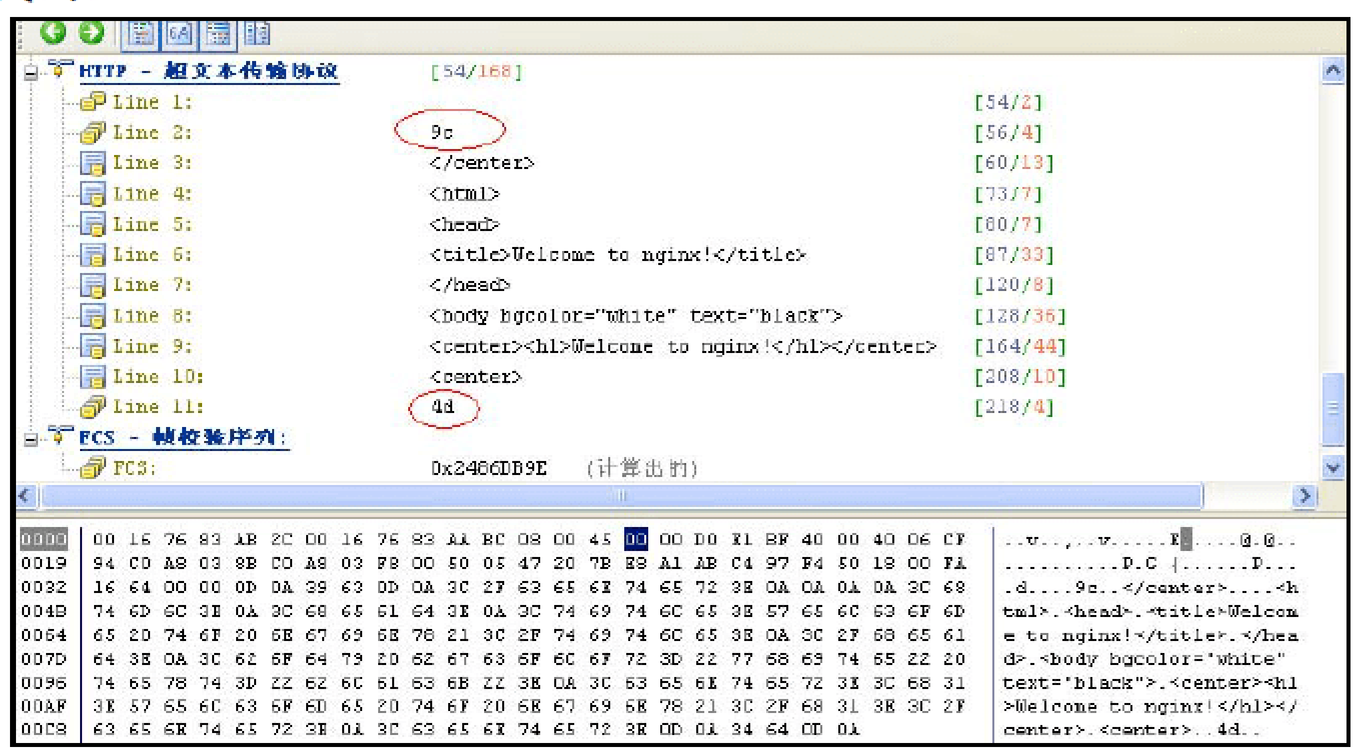
下面是编号为 100 的包：



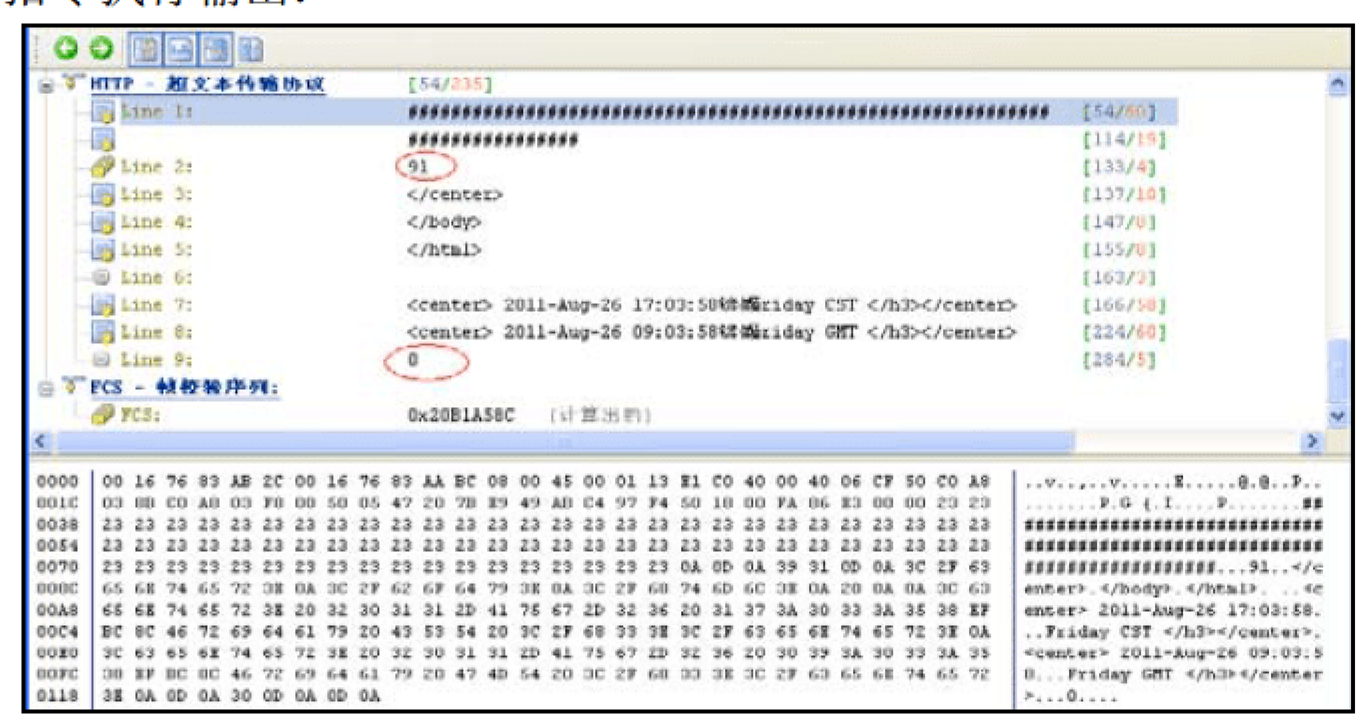
嵌入的 ssi 指令执行输出。我们看到在这个包里仅传输了文件 5.txt 的内容：



部分 html 代码：



嵌入的 ssi 指令执行输出：



可见不是在一个数据包中传输的，而是根据执行情况，在 html 代码中每一个被 SSI 中断处都会有数据包输出。

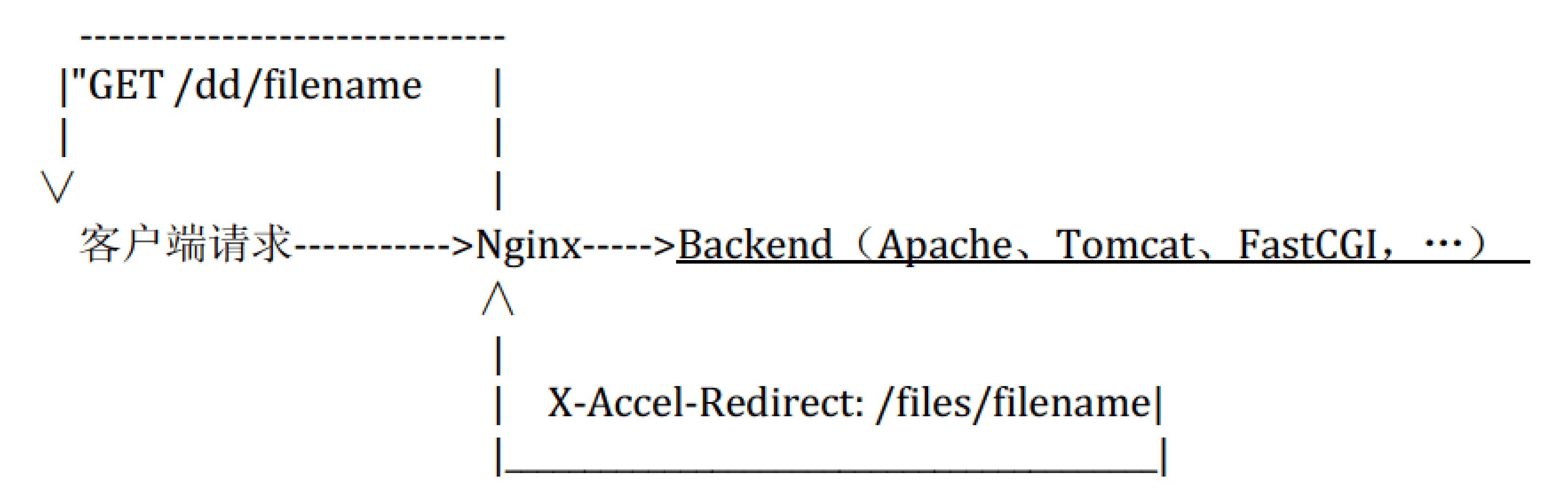
第 25 章　Nginx 与 X-Sendfile

X-accel 模块允许由后台通过返回的头来决定投递静态文件。为什么要这么做呢？试想一下我们经常碰到的这种情况，在有的论坛中，下载资源需要登录认证、权限核查、积分扣除或者是积分增加等等，对于 Nginx 来说这些细粒度的控制其本身无法完成，因此需要应用程序来完成。当应用程序完成这些操作后，根据实际情况会做出选择，如果条件满足那么开始下载所要获取的静态资源（文件）。

注意，这是由动态的程序提供下载，对于动态程序来说，这是一个弱点，而对于 Nginx 来说，这是它的强项。在这种形式下能不能让 Nginx 来完成静态资源的下载呢？答案是可以的。但我们为什么要这么做呢？答案在于 Nginx 在打开静态文件时使用了 sendfile（2），因此其 IO 效率非常高。

25.1 处理流程

具体的处理流程是：



我们从这个流程图中不难看出，在客户端的请求被转向后台服务器时，服务器并没有为客户端返回实际要下载的资源（而是去做了其他的验证或者是其他的工作），而是使用 X-Accel-Redirect 头将下载的资源又传递给了 Nginx，最后又是通过 Nginx 服务器处理该请求发送给客户端。

这种功能就是我们所说的 X-Sendfile，在 Nginx 中由 X-Accel-Redirect 来完成。在后台服务器将下载的请求抛给 Nginx 后，后台服务器又可以承接其他的工作进而处理其他的请求了，因此大大减轻了后端服务器的压力。

相对于其他的 Nginx 来说，X-accel 模块与其他标准的 Nginx 模块有所不同，它的实现不是依赖于指令，而是依赖于在特定方式下后台（或者叫上游）服务器发回的请求头。它的方法我们在前面也了解到了，就是通过发送一个带有 URI 的 X-Accel-Redirect 头，Nginx 将这个请求作为正常（这里的正常是指就像是使用浏览器一样的请求）的请求来处理，然后根据这个 URI 进行 location 匹配，接着请求文件的匹配，最终实现的是在后端服务器返回的请求头中：“root + URI”与 Nginx 中的 location 匹配，这里的“root”，我们以 PHP 程序为例：

```
header（"X-Accel-Redirect: /files/" . $path）;
```

就是我们这个 PHP 程序中的“/files/”部分，而 URI 则是“$path”部分。在此我们就了解到这里，在后面的例子中会证实这一点。

> 另外，还需要注意一点，由于 Nginx 服务器只认识从后端服务器发来的 X-Accel-Redirect 头，而从客户端发来的这种头，它并不理睬。

配置示例

```
# Will serve /var/www/files/myfile.tar.gz
# When passed URI /protected_files/myfile.tar.gz
location /protected files {
 internal;
 alias /var/www/files;
}

# Will serve /var/www/protected_files/myfile.tar.gz
# When passed URI /protected_files/myfile.tar.gz
location /protected files {
 internal;
 root /var/www;
}

You can also proxy to another server.

location /protected_files {
 internal;
 proxy pass http://127.0.0.2;
}
```

25.2 特殊头

在 Nginx 中可以使用以下 5 种头。

头部名称：X-Accel-Redirect

语法：X-Accel-Redirect uri

默认值：X-Accel-Redirect void

功能：为 Nginx 提供访问的 location 设置 URI。

头部名称：X-Accel-Buffering

语法：X-Accel-Buffering [yes|no]

默认值：X-Accel-Buffering yes

功能：为（本）连接设置代理缓存。如果设置为“no”，不允许缓存响应，这种情况适合于 Comet 和 HTTP 流媒体应用；如果设置为“yes”，则允许缓存响应。

Comet：基于 HTTP 长连接的“服务器推”技术，是一种新的 Web 应用架构。基于这种架构开发的应用中，服务器端会主动以异步方式向客户端程序推送数据，而不需要客户端显式地发出请求。Comet 架构非常适合事件驱动的 Web 应用，以及对交互性和实时性要求很强的应用，如股票交易行情分析、聊天室和 Web 版在线游戏等。

服务器推送技术（Server Push）是最近 Web 技术中最热门的一个流行术语，它的别名叫 Comet（彗星）。它是继 AJAX 之后又一个倍受追捧的 Web 技术。服务器推送技术最近的流行与 AJAX 有着密切的关系。

随着 Web 技术的流行，越来越多的应用从原有的 C/S 模式转变为 B/S 模式，享受着 Web 技术 所带来的各种优势（例如跨平台、免客户端维护、跨越防火墙、扩展性好等）。但是基于浏览器的应用，也有它不足的地方，主要在于界面的友好性和交互性。由于浏览器中的页面每次需要全部刷新后才能从服务器端获得最新的数据或向服务器传送数据，这样产生的延迟所带来的视觉感受非常糟糕。因此很多的桌面应用为了获得更友好的界面，放弃了 Web 技术，或者采用浏览器的插件技术（ActiveX、Applet、Flash 等）。但是浏览器插件技术本身又有许多问题，例如跨平台问题和插件版本兼容性问题。

—— 来源于互联网

头部名称：X-Accel-Charset

语法：X-Accel-Charset charset

默认值：X-Accel-Charset utf-8

功能：设置文件的字符集。

头部名称：X-Accel-Expires

语法：X-Accel-Expires [off|seconds]

默认值：X-Accel-Expires off

功能：用于设置 Internal 中的文件在 Nginx 缓存的生存期。

头部名称：X-Accel-Limit-Rate

语法：X-Accel-Limit-Rate bytes [bytes|off]

功能：设置单个请求的速率限制，如果设置为 off，则表示没限制。

25.3 使用实例

在这里我们列举两个例子，一个是服务器间的代理处理；另一个是隐藏真实路径的本机处理。

1. 例 1

在这个配置中，后台服务器的 IP 为 192.168.4.95，Nginx 的 IP 为 192.168.4.98，它们分别位于不同的机器上，客户端访问 IP 为 192.168.4.100：

```
server {
  server name www.xx.com mail.xx.com www.yy.com;
  listen   80;
  root html;
  charset  utf-8;
```

```
 location / {
rewrite ^/download/ (.*)  /d.php?f=$1 last;
proxy pass http://192.168.4.95:80/;
proxy redirect off;
proxy set header   Host $host;
proxy set header   X-Real-IP$remote addr;
proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;

client_max_body_size   10m;
client_body_buffer_size128k;

proxy connect timeout  90;
proxy send timeout 90;
proxy read timeout 90;

proxy_buffer_size  4k;
proxy_buffers  4 32k;
proxy busy buffers size64k;
proxy temp file write size 64k;
  }

 location /files {
 root /t-w/software;
 internal;
 limit_rate_after 3m;
 limit rate 512k;
  }

…
```

编写一个 PHP 文件：

```
[root@jh-share mail]# vi d.php

header ("X-Accel-Redirect: /files/") ;
<?php
// Get requested file name
$path = $_GET["f"];

echo $path;

//…
// Perform any required security checks, validation
```

```
// and/or stats accounting
//…

// And redirect user to internal location
header ("X-Accel-Redirect: /files/" . $path) ;

?>
```

我们在命令行执行这个 PHP 文件：

```
[root@jh-share mail]# php  d.php
PHP Notice:  Undefined index:  f in /mail/httpd/htdocs/mail/d.php on line 3
Content-type: text/html
X-Powered-By: PHP/4.3.9
X-Accel-Redirect: /files/
```

在这里我们要了解的是，请求中添加了 X-Accel-Redirect 头，但是由于没有指定获取的文件，因此变量并没有取到变量值。

> 需要注意的是，该文件是放置在后端服务器中，相应服务器可以访问到的目录中。

↓存在于/t-w/software/files/目录下

现在我们访问: http://www.xx.com**/download/shutnet.bat**

↑虚拟地址用于 Nginx 匹配后重定向

这里的 shutnet.bat 文件是一个内容简单的文件，主要用于说明访问过程。

Nginx 的访问日志：

```
[root@f6 logs]# tail  -f access.log
192.168.4.100- - [28/Aug/2011:12:00:55 +0800] "GET /download/shutnet.bat
HTTP/1.1"  200  4  "-"  "Mozilla/5.0  ( Windows;  U;  Windows  NT  5.1;  en-US;
rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

后台服务器的访问日志：

```
[root@jh-share mail]# tail -f access log
192.168.4.98 - - [28/Aug/2011:11:41:38 +0800] "GET /d.php?f=shutnet.bat
HTTP/1.0" 200 11
```

我们注意到，Nginx 服务器（**192.168.4.98**）在访问后台服务器，它的请求是“**GET /d.php?f=shutnet.bat HTTP/1.0**”，这是由 Nginx 匹配 rewrite 之后的结果，访问状态为 200，由 **d.php** 文件将结果生成后返回 Nginx 服务器，最终由 Nginx 内部处理后再为客户端返回结果。内部处理不记录日志，因此我们看不到处理日志。

如果我们使用 Telnet 方式访问：

```
[root@jh-share ~]# telnet www.xx.com 80
Trying 1.2.3.4…
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET /download/shutnet.bat HTTP/1.1
Host: www.xx.com
```

```
HTTP/1.1 200 OK
Server: nginx/1.0.2
Date: Sun, 28 Aug 2011 04:19:41 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 4
Last-Modified: Sun, 28 Aug 2011 03:11:18 GMT
Connection: keep-alive
Accept-Ranges: bytes

 …
```

我们看到，在输出中是不会有 **X-Accel-Redirect** 头的，因为它是内部头，在后端服务器发往 Nginx 时是有的，而 Nginx 处理该请求之后，再发往客户端的时候就没有该头了。

2. 例 2

在 Nginx 的配置文件中添加以下配置：

```
server {
  server_name www.xx.com;
  listen   80;
  root html;
  charset  utf-8;

  location / {
rewrite ^/download/ (.*)  /d.php?f=$1 last;
fastcgi_pass   127.0.0.1:9001;
fastcgi_index  index.php;
include fastcgi.conf;
   }

  location /files {
 root /usr/local/nginx-1.0.2-mp4-flv/html;
 internal;
 limit_rate_after 1m;
 limit rate 1k;
  }

…
```

例 2 和例 1 相似，不同点在于，在例 2 中动态程序与静态资源位于同一台机器上，因此，我们使用这种方法不但能够减轻后台服务器的压力，还能够隐藏静态资源的真实路径。其他就不再多分析了。

第 26 章　在 Nginx 的响应体之前或之后添加内容

该模块的功能在于能够为当前 location 之前或者之后添加其他 location 的内容，它是作为一个输出过滤模块来执行的，对于主请求的内容（客户端浏览器发送的 URI）和由相当于主 location 访问其他子 location 产生的子请求内容，它们不会被完整地缓冲后再向客户端发送，而是以数据流的形式发往客户端。由于在向客户端发送数据时总的内容长度并不知道，因此在发往客户端时不会有 Content-Length 头，而是采用了 HTTP chunked 编码的方式来实现动态提供请求体（body）的长度，因此，在使用该模块时总是会使用 chunked 编码。

该模块在 Nginx 的默认安装方式下是没有被安装的，要想使用该模块，那么在编译安装 Nginx 时需要添加--with-http_addition_module 选项，例如：

```
[root@web6  nginx-1.0.2]#./configure  --prefix=/usr/local/nginx-1.0.2-addition  --with-http_addition_module
```

1. 配置示例

```
location / {
  add_before_body   /before_action;
  add_after_body/after_action;
}
```

需要注意的一点是在指令 add_after_body 或 add_before_body 中不能使用变量，例如以下格式为错误的使用方法：

```
location / {
  set $before action /before action;
  add_before_body $before_action;
}
```

2. 指令

该模块提供了 3 条指令。

指令名称：add_before_body

语法：add_before_body uri

默认值：no

使用环境：http，server，location

功能：该指令用于在响应主体内容之前添加 URI 子请求内容，作为一个子请求的结果将请求的内容添加在请求主体之前。

指令名称：add_after_body

语法：add_after_body uri

默认值：no

使用环境：http，server，location

功能：该指令用于在响应主体内容之后添加 URI 子请求内容，作为一个子请求的结果将请求

的内容添加在请求主体之后。

指令名称：addition_types

语法：addition_types mime-type [mime-type ...]

默认值：text/html

使用环境：http，server，location

功能：该指令从 0.7.9 版本之后提供使用，它用于在 location 中指定 MIME 类型 （默认为"text/html"）。

在 Nginx 0.8.17 版本之前，这个指令被错误地写为"addtion_types"，在 0.8.17 版本中被修复，这一点需要注意。

3. 使用实例

在这里我们举了两个例子，分别用于说明不同配置情况下的结果。

例 1

在这个例子中，不同 location 的 root 在系统中的物理路径不同。

```
server {
  listen   80;
  server name  www.xx.com;

  location / {
root   html;
index  index.html index.htm;
add_before_body /header/;
add_after_body  /footer/;

  }

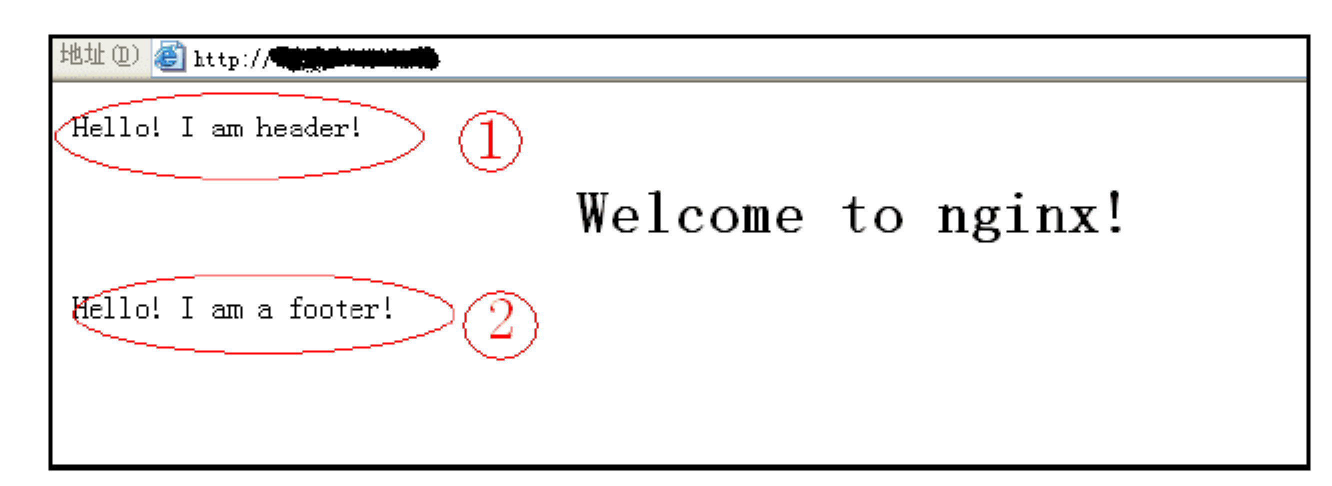
  location /header {
 root   /www;
 index  index.html index.htm;
  }

  location /footer {
 root   /var/www;
 index  index.html index.htm;
  }

...

}
```

我们来访问 http://www.xx.com，截图如下：



在这个截图中：①处的内容来自于/www/header/index.html 文件，而②处的内容来自于/var/www/footer/index.html 文件。

查看 Nginx 的访问日志：

```
[root@mail logs]# tail  -f access.log
192.168.3.248 - - [28/Aug/2011:15:57:35 +0800] "GET / HTTP/1.1" 200 260 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
```

最终呈现在我们面前的页面是由三个页面组成的。而在 Nginx 的访问日志中却只有一条日志，因为其他两个文件的访问是由 Nginx 内部访问完成的。

下面我们以 Telnet 方式访问：

```
[root@jh-share ~]# telnet www.xx.com80
Trying 1.2.3.4...
Connected to www.xx.com (1.2.3.4).
Escape character is '^]'.
GET / HTTP/1.1
Host: www.xx.com

HTTP/1.1 200 OK
Server: nginx/1.0.2
Date: Sun, 28 Aug 2011 08:25:23 GMT
Content-Type: text/html
Last-Modified: Sun, 28 Aug 2011 05:28:04 GMT
Transfer-Encoding: chunked
Connection: keep-alive

29
 Hello! I am header!

97
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>Welcome to nginx!</h1></center>
</body>
```

```
</html>

2d
   Hello! I am a footer!

0

Connection closed by foreign host.
```

从这个访问结果中，我们能够看出使用的是“**Transfer-Encoding: chunked**”方式，注意结果中的数字，它是以十六进制方式表示，表明从某处到另一处的字节大小，这就是 chunked 方式保证数据传输的要点所在。

例 2

在 Nginx 的配置文件中添加以下内容：

```
server {
  listen   80;
  server_name  www.xx.com;

  location / {
root   html;
index  index.html index.htm;
add_before_body /header/start.html;
add_after_body  /footer/end.html;

  }
}
```

我们再来访问 http://www.xx.com，截图如下：

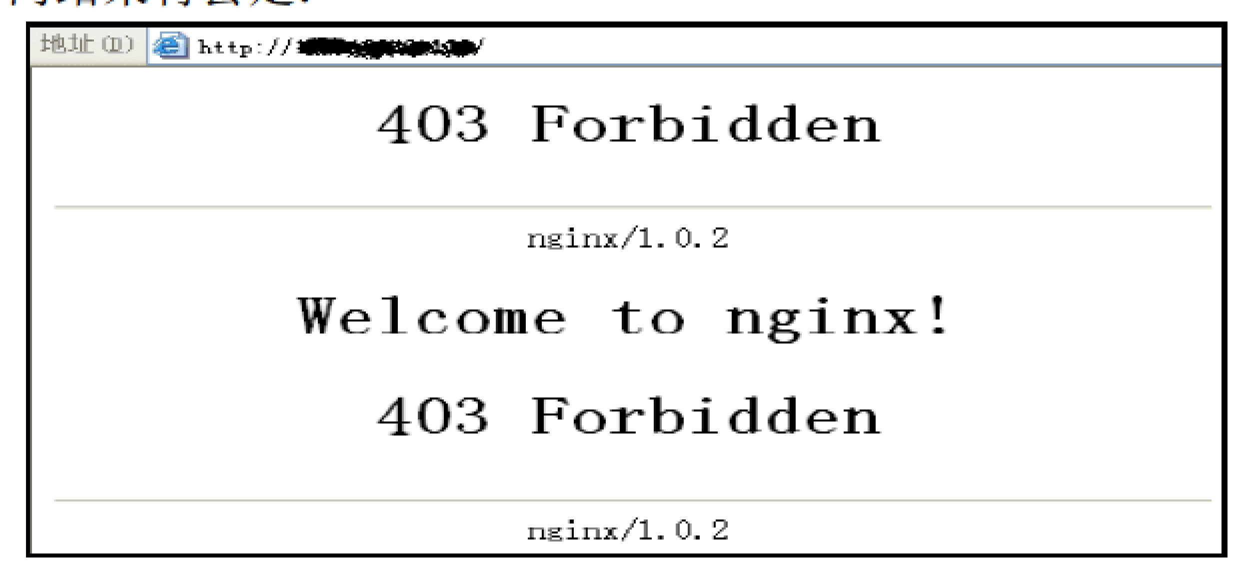


在这里，我们要访问的 **/header/start.html** 和 **/footer/end.html** 都在同一个“root html”的管辖范围内，因此就不用再添加其他的 location 了。另外，我们还注意到，在这个例子中由于没有使用 index.html 格式，因此就使用了全路径指定文件。如果将 Nginx 的配置文件写为以下这种格式：

```
  location / {
root   html;
index  index.html index.htm;
add_before_body /header/;
```

```
  add_after_body  /footer/;
  }
```

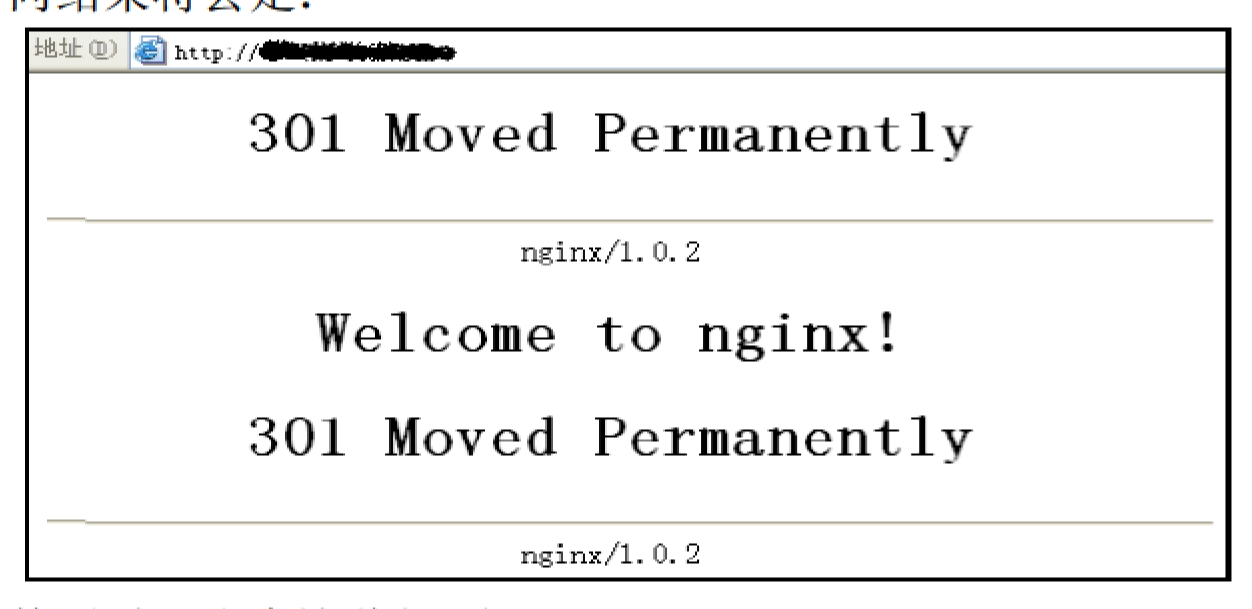
那么我们的访问结果将会是：



还有一种情况，如果我们在配置文件中写为：

```
  location / {
root   html;
index  index.html index.htm;
add before body /header;
add after body  /footer;
  }
```

那么我们的访问结果将会是：



因此，在具体使用时要注意这些问题。

第 27 章　Nginx 与访问者的地理信息

Nginx 可以通过配置使用 http_geoip_module 模块来记录、使用访问者的信息，或者是根据这些信息有选择地提供服务。

http_geoip_module 模块会创建一些 ngx_http_geoip_module 变量，这些编码是基于客户端 IP 的，它们会与 MaxMind GeoIP 二进制文件进行匹配查询。该模块仅用于 0.7.63 和 0.8.6 版本之后的版本中。

另外，http_geoip_module 模块需要 geo 数据库和读取数据库的库文件，也就是说 Nginx 本身并不提供这些数据库和库文件，因此，我们需要另外的下载和安装。

Nginx 的默认安装并不包括 http_geoip_module 模块，因此，要想使用该模块，需要在安装 Nginx 时指定--with-http_geoip_module 选项。

1. 下载安装

下载并解压相关数据库：GeoIP.dat 和 GeoLiteCity.dat：

```
[root@mail ~]# wget http://geolite.maxmind.com/download/ \
> geoip/database/GeoLiteCountry/GeoIP.dat.gz
[root@mail ~]#gzip -d GeoIP.dat.gz
[root@mail ~]# wget http://geolite.maxmind.com/download/\
> geoip/database/GeoLiteCity.dat.gz
[root@mail ~]#gzip -d GeoLiteCity.dat.gz
```

编译安装 Nginx：

```
[root@mail nginx-1.0.2]# ./configure --prefix=/usr/local/nginx-1.0.2-Geo
--with-http_geoip_module

…

./configure: error: the GeoIP module requires the GeoIP library.
You can either do not enable the module or install the library.
```

如果在安装 Nginx 的过程中出现了以上错误，那么需要安装以下软件：

```
[root@mail ~]# wget http://geolite.maxmind.com/download/geoip/api/c
/GeoIP.tar.gz
[root@mail ~]# tar -zxvf GeoIP.tar.gz
[root@mail ~]#cd GeoIP-1.4.8/
[root@mail GeoIP-1.4.8]#./configure
[root@mail GeoIP-1.4.8]# make
[root@mail GeoIP-1.4.8]# make check
[root@mail GeoIP-1.4.8]# make install
```

也就是说需要某些库文件。然后再接着安装 Nginx，安装完成 Nginx 后，可以通过以下命令来检测 Nginx 是否具备了 http_geoip_modul 模块：

```
[root@mail conf]# /usr/local/nginx-1.0.2/sbin/nginx  -V
nginx: nginx version: nginx/1.0.2
nginx: built by gcc 4.1.2 20070626 (Red Hat 4.1.2-14)
nginx: configure arguments: --prefix=/usr/local/nginx-1.0.2
--with-http_geoip_module
```

最后一行告诉我们，该模块已经安装成功了。

2. 配置示例

```
http {
geoip_country  GeoIP.dat;  # the country IP database
geoip city GeoLiteCity.dat; # the city IP database
 (...)
```

使用配置：

```
fastcgi_param GEOIP_COUNTRY_CODE $geoip_country_code;
fastcgi_param GEOIP_COUNTRY_NAME $geoip_country_name;
```

3. 指令

Nginx 将数据库缓存到内存中，对于 IP 对应国家的数据库很小，最新 GeoIP-1.4.8 提供的 GeoIP.dat 大约只有 1.2MB，因此，它并不会占用多大内存，但是城市的数据库很大，约 18MB，因此会带来更大的内存占用。

指令名称：geoip_country

语法：geoip_country path/to/db.dat

默认值：none

使用环境：http

功能：该指令用于指定 Geo IP 数据库.dat 的全路径，该数据库包含了 IP 地址与国家的关系，就是说可以通过给定 IP 地址获取该 IP 所在的国家。它的功能在于能够通过客户端的房屋地址来决定访问者所在的国家。如果使用了该指令，那么可以使用以下变量。

- $geoip_country_code：两个字母的国家代码。例如，“RU”，“US”；
- $geoip_country_code3：三个字母的国家代码。例如，“RUS”，“USA”；
- $geoip_country_name：国家全名称，“Russian Federation”，“United States”。

指令名称：geoip_city

语法：geoip_city path/to/db.dat

默认值：none

使用环境：http

功能：该指令用于指定 Geo IP 数据库.dat 的全路径，该数据库包含了 IP 地址与国家、城市和区域名称、地理位置（经度、纬度）的关系，就是说对于给定的 IP 地址可以获取相应的信息。它的功能在于通过 IP 地址来决定访问者的地理坐标信息。如果使用了该指令，那么可以使用以下变量。

- $geoip_city_country_code：两个字母的国家代码。例如，“RU”，“US”；
- $geoip_city_country_code3 ：三个字母的国家代码。例如，“RUS”，“USA”；
- $geoip_city_country_name ：国家全名称，“Russian Federation”，“United States”；
- $geoip_region：区域名称（省份、地区、州、行政区域等），例如，“oscow City”，“DC”；
- $geoip_city：城市的名字。例如，“Moscow”，“Washington”，“Lisbon”，&c ；$geoip_postal_code：邮递区号或者邮政编码；
- $geoip_city_continent_code：大陆代码；
- $geoip_latitude ：纬度；
- $geoip_longitude ：经度。

4. 配置实例

在这里我们举了三个例子，其中例 1 和例 3，我们在具体的使用中可以应用，例 2 只是用于说明该模块功能，但是如果你想做登录信息提示，或者是将该信息记录在数据库中，那么例 2 将是一个很好的应用。

例 1

在 Nginx 服务器的配置文件中添加以下配置内容：

```
http {
include   mime.types;
default type  application/octet-stream;
sendfileon;

keepalive_timeout  65;

geoip_country  /usr/local/nginx-1.0.2/conf/GeoIP.dat;
geoip_city /usr/local/nginx-1.0.2/conf/GeoLiteCity.dat;

log_format custom $time_local | $server_name |$request_length | $bytes_sent
| --$geoip_country_name--$geoip_city--$geoip_postal_code;

server {
  listen   80;
  server_name  www.xx.com;

  location / {
root   html;
index  index.html index.htm;
access_log logs/custom.log custom;
  }
}
```

```
…
}
```

查看访问日志：

```
[root@mail logs]# tail -f custom.log
29/Aug/2011:09:53:38 +0800|localhost|614|366|------
29/Aug/2011:09:55:58 +0800|localhost|319|157|------
29/Aug/2011:10:12:45 +0800| www.xx.com |565|366|--United States----
29/Aug/2011:13:08:54 +0800| www.xx.com |320|157|------
29/Aug/2011:13:14:14 +0800| www.xx.com |565|366|----Beijing--
29/Aug/2011:13:19:16 +0800| www.xx.com |270|366|--China--Beijing--
29/Aug/2011:13:19:22 +0800| www.xx.com |320|157|--China--Beijing--
29/Aug/2011:13:25:56 +0800| www.xx.com |270|366|--China--Beijing--
```

例 2

在 Nginx 的配置文件中添加以下配置内容：

```
http {
  include   mime.types;
  default type  application/octet-stream;

  sendfileon;
  keepalive_timeout  65;

geoip_country  /usr/local/nginx-1.0.2/conf/GeoIP.dat;
geoip_city /usr/local/nginx-1.0.2/conf/GeoLiteCity.dat;

  server {
listen   80;
server_name  www.xx.com;

location / {
  root   html;
  index  index.html index.htm;
}

location ~ .php$ {
  fastcgi_pass   192.168.4.156:9001;
  fastcgi_index  index.php;
  include fastcgi.conf;
  fastcgi_param GEOIP_COUNTRY_CODE3 $geoip_country_code3;
  fastcgi_param GEOIP_COUNTRY_NAME $geoip_country_name;

  fastcgi_param GEOIP_CITY_COUNTRY_CODE $geoip_city_country_code;
```

```
    fastcgi_param GEOIP_CITY_COUNTRY_CODE3 $geoip_city_country_code3;
    fastcgi_param GEOIP_CITY_COUNTRY_NAME $geoip_city_country_name;
    fastcgi_param GEOIP_REGION $geoip_region;
    fastcgi_param GEOIP_CITY $geoip_city;
    fastcgi_param GEOIP_POSTAL_CODE $geoip_postal_code;
    fastcgi_param GEOIP_CITY_CONTINENT_CODE $geoip_city_continent_code;
    fastcgi_param GEOIP_LATITUDE $geoip_latitude;
    fastcgi_param GEOIP_LONGITUDE $geoip_longitude;
    }
  }
```

添加两个.php 文件，一个用于获取国家信息，另一个用于获取城市信息，这两个文件来源于互联网，非本人编写。

用于获取国家信息的.php 文件：

```
[root@mail html]# vi geoip.php

<html>
<head>
  <title>What is my IP address - determine or retrieve my IP address</title>
 </head>
<body>
 <?php
 if (getenv(HTTP_X_FORWARDED_FOR)) {
$pipaddress = getenv(HTTP_X_FORWARDED_FOR);
$ipaddress = getenv(REMOTE_ADDR);
echo "Your Proxy IP address is : ".$pipaddress. " (via $ipaddress) " ;
} else {
$ipaddress = getenv(REMOTE_ADDR);
echo "Your IP address is : $ipaddress";
}
$country = getenv(GEOIP_COUNTRY_NAME);
$country_code = getenv(GEOIP_COUNTRY_CODE);
echo "<br/>Your country : $country ( $country_code ) ";
?>
</body>
</html>
```

访问测试：

地址(D) http://[redacted]geoip.php

Your IP address is : 61.135.169.106
Your country : China ( )

用于获取城市信息的.php 文件：

```
[root@mail html]# vi geoip_city.php

<html>
<head>
 <title>What is my IP address - determine or retrieve my IP address</title>
</head>
<body>
<?php
if (getenv(HTTP_X_FORWARDED_FOR)) {
$pipaddress = getenv(HTTP_X_FORWARDED_FOR);
$ipaddress = getenv(REMOTE_ADDR);
echo "<br>Your Proxy IP address is : ".$pipaddress. " (via $ipaddress) " ;
} else {
$ipaddress = getenv(REMOTE_ADDR);
echo "<br>Your IP address is : $ipaddress";
}
  $geoip_city_country_code = getenv(GEOIP_CITY_COUNTRY_CODE);
  $geoip_city_country_code3 = getenv(GEOIP_CITY_COUNTRY_CODE3);
  $geoip_city_country_name = getenv(GEOIP_CITY_COUNTRY_NAME);
  $geoip_region = getenv(GEOIP_REGION);
  $geoip_city = getenv(GEOIP_CITY);
  $geoip_postal_code = getenv(GEOIP_POSTAL_CODE);
  $geoip_city_continent_code = getenv(GEOIP_CITY_CONTINENT_CODE);
  $geoip_latitude = getenv(GEOIP_LATITUDE);
  $geoip_longitude = getenv(GEOIP_LONGITUDE);
  echo "<br>Country : $geoip_city_country_name ( $geoip_city_country
code3 , geoip_city_country_code ) ";
  echo "<br>Region :  $geoip_region";
  echo "<br>City :  $geoip_city ";
  echo "<br>Postal code :  $geoip_postal_code";
  echo "<br>City continent code :  $geoip_city_continent_code";
  echo "<br>Geoip latitude :  $geoip_latitude ";
  echo "<br>Geoip longitude :   $geoip_longitude ";

?>
</body>
</html>
```

访问测试：

```
地址(D) http://[redacted]/geoip_city.php

Your IP address is : 61.135.169.106
Country : China ( CHN , CN )
Region : 22
City : Beijing
Postal code :
City continent code : AS
Geoip latitude : 39.9289
Geoip longitude : 116.3883
```

例 3

在 Nginx 的配置文件中添加以下配置：

```
http {

 ...

geoip_country  /usr/local/nginx-1.0.2/conf/GeoIP.dat;
geoip_city /usr/local/nginx-1.0.2/conf/GeoLiteCity.dat;

  server {
listen  80;
server_name  www.xx.com;
charset utf-8;

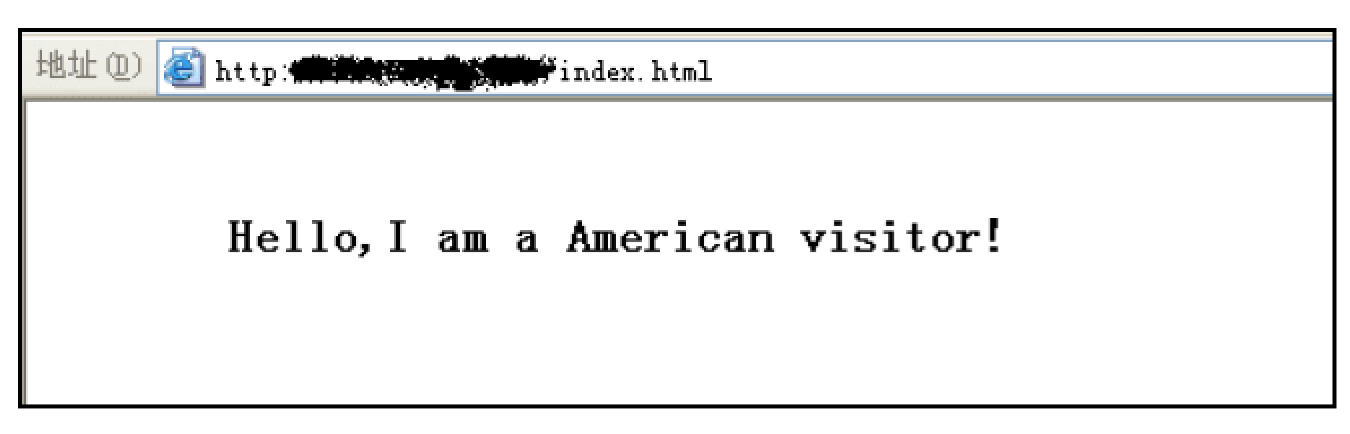
location / {
  root   /var/www/html;
  index  index.html index.htm;

  if  ($geoip_country_code = US)  {
 rewrite  ^(.*)$  /us/$1  break;
  }
  if  ($geoip_country_code = CN)  {
 rewrite  ^(.*)$  /cn/$1  break;
  }
  if  ($geoip_country_code = '' )  {
 rewrite  ^(.*)$  /cn/$1  break;
  }
}

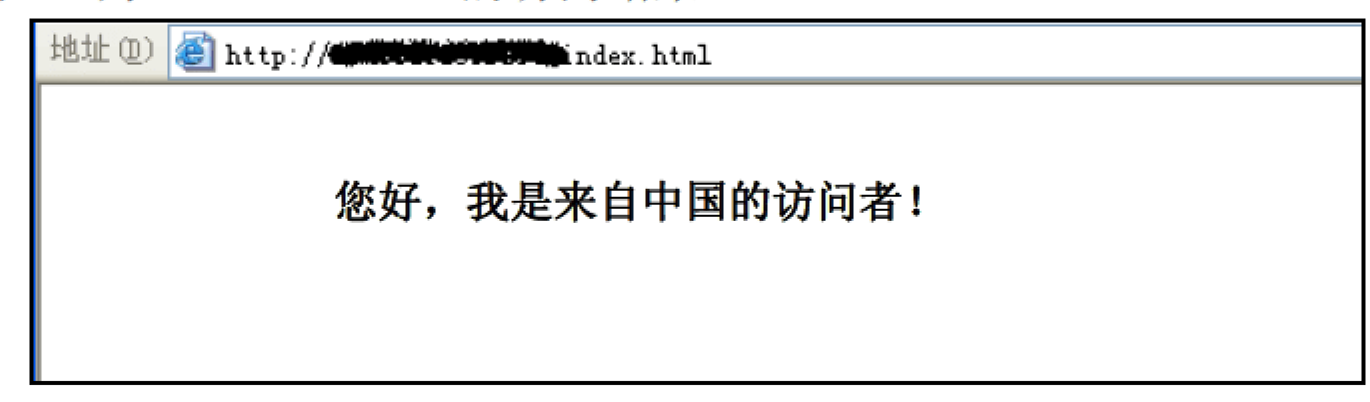
 ...
 }
```

访问测试

这是来自于 IP 为 23.123.123.125 的访问结果：



这是来自于 IP 为 219.237.47.xx 的访问结果：



配置文件中的最后一种情况，是如果出现国家代码无法找到时，提供的默认访问。

第 28 章　Nginx 的图像处理

在 Nginx 服务器上可以对某些图像进行处理，这个功能是由一个叫做 http_image_filter_module 的模块来实现的。

该模块是一个过滤器，主要用来裁剪过大的图片，在 0.7.54 以上的 Nginx 版本中提供了这个功能模块，用于对被传递的 JPEG, GIF 和 PNG 图像进行一些改变。由于该模块在默认安装中没有选择，因此，要想使用该模块，那么在进行./configure 时需添加--with-http_image_filter_module 选项。

> 另外，还需要注意的是，该模块依赖于 libgd，因此我们首先需要安装 libgd，它的官方下载地址为 https://bitbucket.org/pierrejoye/gd-libgd，在此我们就不再讲述安装了。

1. 配置示例

```
location /img/ {
proxy pass http://backend;
image filter   resize  150 100;
error_page 415   = /empty;
}

location = /empty {
empty gif;
}
```

2. 指令

http_image_filter_module 模块提供了以下 4 条指令。

指令名称：image_filter

语法：image_filter （test|size|resize width height|crop width height）

默认值：none

使用环境：location

功能：该指令用于指定应用到图像的转换类型。可选择的类型如下：

- test：用于检验返回响应中的图像确实是 PEG、GIF 或 PNG 格式，否则将返回 415 错误。
- size：通过 JSON 格式给定图像的信息，例如：

```
{ "img" : { "width": 100, "height": 100, "type": "gif" } }
```

如果发生错误，那么将是：

```
{}
```

- resize：按比例将图像缩小到指定的大小。
- crop：按比例将图像缩小到指定的大小，并且会裁剪多余的边缘。

指令名称：image_filter_buffer

语法：image_filter_buffer size

默认值：1M

使用环境：http，server，location

功能：设置用于读取图片的最大缓存值，这个值的设定对于图像传输速度影响很大。

指令名称：image_filter_jpeg_quality

语法：image_filter_jpeg_quality [0...100]

默认值：75

使用环境：http，server，location

功能：该指令用于设置处理 JPEG 图片时损失的质量比率，也叫品质，最大推荐值为 95。

指令名称：image_filter_transparency

语法：image_filter_transparency on|off

默认值：on

使用环境：http，server，location

功能：该指令用于设置对 GIF 格式文件和基于调试版的 PNG 格式文件禁用图像透明，以便提高图像的重采样质量。对于真彩 PNG alpha 通道总是被保留，而不管是否有这个设置。注意：灰度 PNG 并未经过测试，但是它将应该作为真色彩进行处理。

3. 使用配置

例 1

在 Nginx 中添加以下配置：

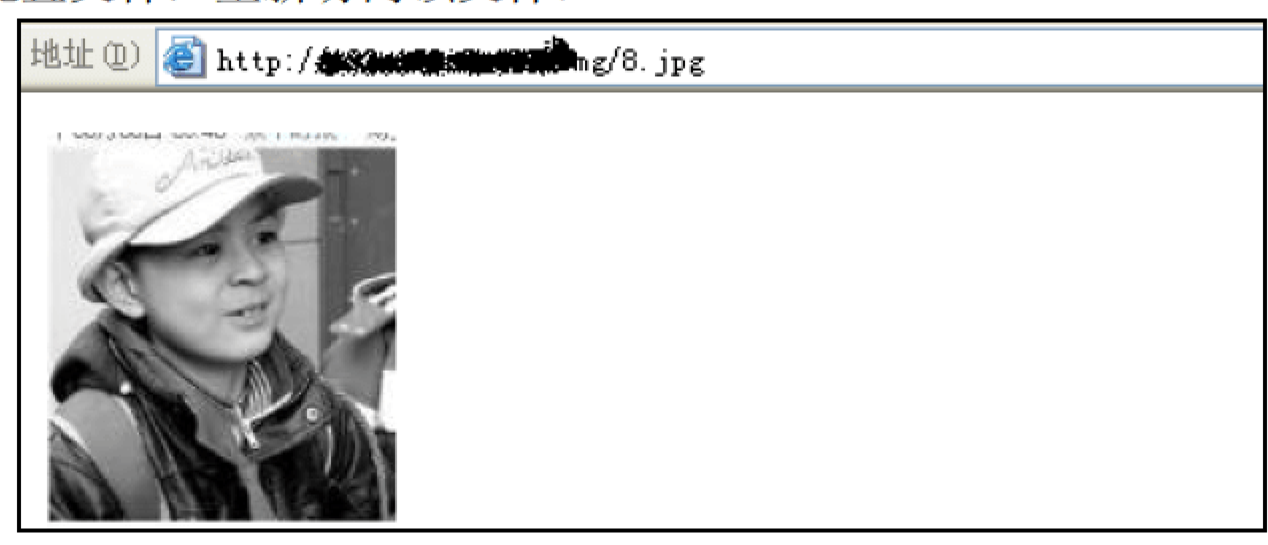
```
location /img/ {
proxy_pass http://192.168.10.18:80/;
image_filter_buffer 500k;
image_filter   resize  200 150;
error_page 415   = /empty;
}

location = /empty {
empty_gif;
}
```

在没有重新载入配置之前，先访问以下图像文件，看看实际的大小：



然后再载入配置文件，重新访问该文件：



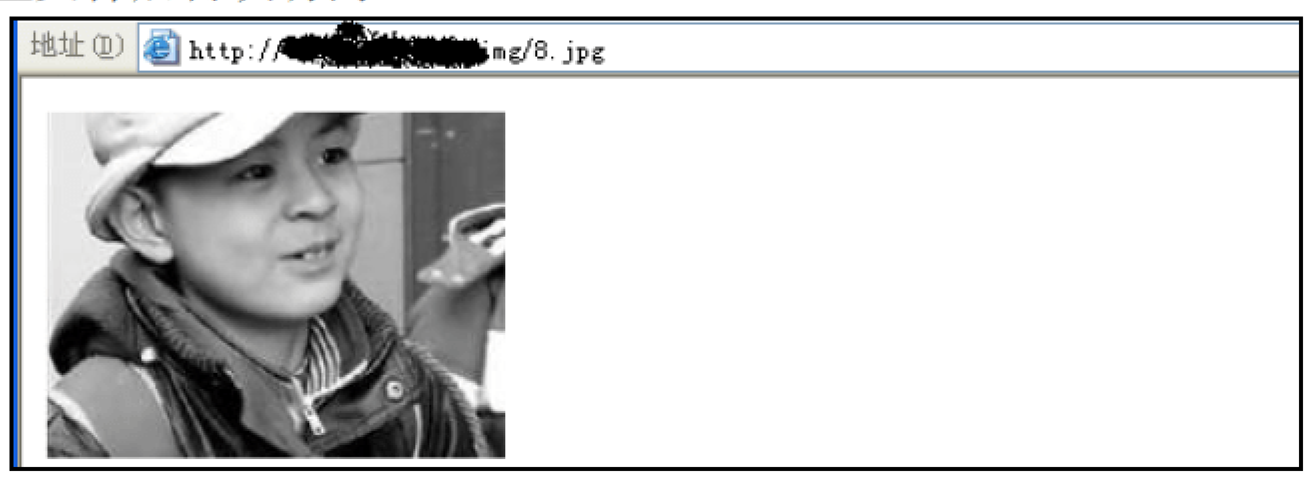
例 2

在 Nginx 的配置文件中添加以下内容：

```
location /img/ {
proxy pass http://192.168.3.140:80/;
image filter buffer 500k;
image filter  crop  200 150;
error_page 415   = /empty;
}

location = /empty {
empty gif;
}
```

重新载入配置文件后再次访问：



第 29 章　location 中随机显示文件

如果想在众多的文件中随机选择一个文件，无论是 html 文件还是图像文件都是可以的，如果将图像文件作为一种随机显示，那么这个功能确实不错。要启用该功能，在编译 Nginx 时需要添加--with-http_random_index_module 选项才能启用，在默认安装 Nginx 中是没有安装该模块的。

1. 配置示例

```
location  /  {
  random_index  on;
}
```

2. 指令

该模块仅提供了一条指令，那就是用开启这个功能。

指令名称：random_index

语法：random_index [on|off]

默认值：off

使用环境：location

功能：如果在一个指定的 location 中使用该指令，那么 Nginx 将会扫描给定目录中的文件，对于每一次的访问都会随机给出一个文件而不是通常的 index.html 文件。可以是文本文件（html、txt 等）、图像文件（JPG、BMP、GIF 和 PNG）等文件类型。需要注意的是，文件名不能以 “.” 号开头，这种文件不会被读取。

3. 使用配置

在 Nginx 的配置文件中添加以下配置：

```
location /tx1 {
   root  /www/image
   random index  on;
}
```

我们看一下目录/www/image/tx1 的结构：

```
[root@mail tx1]# tree
.
|-- 1.gif
|-- 2.gif
…
'-- v2
|-- 1.gif
'-- 2.gif
```

```
    1 directory, 160 files
```

当我们访问 http://www.xx.com/tx1 时，将会以随机的方式显示头像图片，而不会显示出 v2 目录下的文件，就是说只在本机目录搜索；如果我们访问 http://www.xx.com/tx1/v2 时，那么 v2 目录中的图像同样被随机显示，这说明“random_index on”在同一个 location 中是可以继承的。

第 30 章　后台 Nginx 服务器记录原始客户端的 IP 地址

通过这个模块允许我们改变客户端请求头中客户端 IP 地址值（例如，X-Real-IP 或 X-Forwarded-For）。

如果 Nginx 工作在某些 7 层负载均衡代理后面，该功能对于 Nginx 服务器非常有用，由于客户端请求的本地 IP（就是客户端的请求地址）在通过 7 层代理时被添加了客户端 IP 地址头，因此才使得后端的 Nginx 能够取得震慑客户端的 IP 地址值。该模块在默认安装时并没有安装，因此如果要使用该模块，那么在编译安装时需要添加--with-http_realip_module 选项。

为什么使用该模块，它的意义在于能够使得后台服务器记录原始客户端的 IP 地址。

1. 配置示例

```
set real ip from  192.168.1.0/24;
set real ip from  192.168.2.1;
real_ip_header X-Real-IP;
```

2. 指令

该模块仅提供了两条指令。

指令名称：set_real_ip_from

语法：　set_real_ip_from [the address|CIDR|"unix:"]

默认值：none

使用环境：http，server，location

功能：通过该指令指定信任的地址，将会被替代为精确的 IP 地址。从 0.8.22 版本后也可以使用信任的 UNIX 套接字。这里设置的 IP 就是指前端 Nginx、Varnish 或者 Squid 的 IP 地址。

指令名称：real_ip_header

语法：real_ip_header [X-Real-IP|X-Forwarded-For]

默认值：real_ip_header X-Real-IP

使用环境：http，server，location

功能：该指令用于设置使用哪个头来替换 IP 地址。如果使用了 X-Forwarded-For，那么该模块将会使用 X-Forwarded-For 头中的最后一个 IP 地址来替换前端代理的 IP 地址。

3. 使用实例

在下面的实例中，我们的环境是这样的：有两台 Nginx 服务器，一台是前端，另一台是后端，前端的 Nginx 被用做代理，而后端的 Nginx 用于提供页面访问，还有一台客户端，IP 地址如下：

- 前端 Nginx：192.168.7.10
- 后端 Nginx：192.168.1.15
- 客户端主机：218.239.201.36

前端的 Nginx 配置如下。

```
    server {
      listen   80;
      server_name  www.xx.com;

  location / {
    root   html;
    index  index.html index.htm;
    charset   utf-8;
      }

  location  /865 {
    proxy pass   http://192.168.3.139:80/;
    proxy set header  X-Real-IP$remote addr;
    proxy set header  X-Forwarded-For  $proxy add x forwarded for;
    proxy set header  Host $host;
    proxy_redirect off;
      }

    …
    }
```

后端的 Nginx 配置如下：

```
    server {
      listen   80;
      server name  localhost;

      location / {
    root   /var/www/html;
    index  index.html index.htm;
      }
```

访问测试

如果我们访问 http://www.xx.com/865，没问题，可以是正常访问。访问日志如下。

前端 Nginx 的日志：

```
    218.239.201.36 - - [30/Aug/2011:16:09:56 +0800] "GET /865/ HTTP/1.1" 200
151 "-" "Mozilla/5.0 （Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12）
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

后端 Nginx 的日志：

```
    192.168.7.10 - - [30/Aug/2011:16:09:56 +0800] "GET // HTTP/1.0" 200 151 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026
Firefox/3.6.12 GTB7.1"
```

我们看到在后端 Nginx 的日志中并没有记录原始客户端的 IP 地址，而是记录了前端 Nginx

的 IP 地址。

如果将后台 Nginx 服务器的配置修改为：

```
server {
  listen   80;
  server_name  localhost;
  set_real_ip_from   192.168.3.0/24;
set_real_ip_from   100.100.0.0/16;
  real_ip_header X-Real-IP;

  location / {
root   html;
index  index.html index.htm;
  }

...
}
```

然后我们再次进行访问测试。

前端 Nginx 的日志：

```
218.239.201.36 - - [30/Aug/2011:16:10:28 +0800 "GET /865/ HTTP/1.1" 304 0 "-" "Mozilla/5.0 ( Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12 ) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

后端 Nginx 的日志：

```
218.239.201.36 - - [30/Aug/2011:16:10:28 +0800] "GET // HTTP/1.0" 304 0 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

可见，这次后台记录的是客户端的 IP 地址。

第 31 章　解决防盗链

首先明白一下什么是盗链接:

盗链是指服务提供商自己不提供服务的内容，通过技术手段绕过其他有利益的最终用户界面（如广告），直接在自己的网站上向最终用户提供其他服务提供商的服务内容，骗取最终用户的浏览和点击率。受益者不提供资源或提供很少的资源，而真正的服务提供商却得不到任何的收益。

盗链形式的不同，可以简单地把盗链接分成两类：常规盗链接和分布式盗链。常规盗链接比较初级，同时也比较常见，具有一定的针对性，只盗用某个或某些网站的链接。技术含量不高，实现也比较简单，只需要在自己的页面嵌入别人的链接即可。分布式盗链接是盗链的一种新的形式，系统设计复杂，难度相对较大。这种盗链接一般不针对某一个网站，互联网上任何一台机器都有可能成为盗链接的对象。服务提供商一般会在后台设置专门程序（Spider）在 Internet 上抓取有用的链接，然后存储到自己的数据库中。而对于最终用户的每次访问，都将其转化为对已有数据库的查询，被查询到的 URL 就是被盗链的对象。由于对文件的访问已经被浏览器屏蔽掉了，所以最终用户感觉不到所访问的链接是被盗取的链接。

—— 来自互联网

可见，如果你的网站被盗链后，你就会变成一个为他人服务的机器了（你好无私呀！）。那么在 Nginx 下怎么解决这个问题呢？

在这里我们将会了解到可以使用三种方法解决盗链，即使用 Referer 模块、AccessKey 模块或者是 Secure Link 模块。

31.1 使用 Referer 模块

Referer 模块提供了一条指令 valid_referers，它的目的是检查来自于客户端请求的 Referer HTTP 头，并且可能会拒绝基于该值的请求。如果 Referer 被认为无效，那么$invalid_referer 设置为 1。

要知道欺骗一个 Referer HTTP 头是非常简单的过程，因此检查客户端请求的 Referer 并不能作为一个安全措施，只能防止一部分这类人的行为，所以使用该模块不能 100%地阻止盗链请求。

1. 配置示例

我们看一下官方给的一个例子：

```
location /photos/ {
  valid referers none blocked www.mydomain.com mydomain.com;

  if ($invalid_referer) {
return  403;
  }
}
```

指令 valid_referers 会对来源进行判断，如果不属于它的值，那么对于变量$invalid_referer，如果发现 Referer 无效，则返回一个错误代码。

2. 指令

该模块仅提供了一条指令— valid_referers、一个变量$invalid_referer。

指令名称：valid_referers

语法：valid_referers [none|blocked|server_names] ...

默认值：no

使用环境：server，location

功能：该指令会根据 Referer 头来指定$invalid_referer 变量的值，0 或者 1，可以通过该指令来实现防止盗链，如果 Referer 头没有出现在由 valid_referers 指令指定的列表中，那么变量$invalid_referer 的值为 1。

相关参数如下：

- none：“Referer”头缺席（absence）被认为是有效的；
- blocked：由防火墙伪装的 Referrer ，例如“Referer: XXXXXXX”也被认为是有效的；
- server_names：被指定的服务器名字被认为是有效的 Referer。它的值是一个列表，可以列举一个或者多个服务器，在 Nginx 的 0.5.33 版本之后，可以在服务器名字中使用*号。

3. 使用实例

我们看下面的配置：

```
location ~* \.(gif|jpg|png|bmp|swf|flv|mp4|mp3)$ {
      valid referers none blocked www.t1.com www.t2.com;
      if ($invalid_referer) {
      rewrite ^/ http://www.t1.com/403.html;
      }
}
```

在该配置中，对于图片、视频及音乐格式的文件而言禁止盗链。

由于这个模块不是很有用，因此在这里就不再举例了。

31.2 使用 AccessKey 模块

下面认识一个新模块——HttpAccessKeyModule，该模块没有在 Nginx 的发布中包含。

该模块会拒绝用户访问，除非在请求的 URL 中包含访问 key，key 可能是远程的 IP 地址或者是其他的变量，因此这样可以限制某些用户进行动态下载。换句话说就是使用了“访问令牌”，这样不知道令牌的人则无法访问，因此可以很好地控制客户端的下载行为。

1. 配置示例

```
location /download {
  accesskey on;
```

```
    accesskey_hashmethod  md5;
    accesskey_arg "key";
    accesskey_signature   "mypass$remote_addr";
  }
```

客户端可能会指向，例如：

```
http://example.com/download/file.zip?key=09093abeac094
```

2. 安装模块

首先下载源代码 ginx-accesskey-2.0.3.tar.gz：

```
[root@mail ~]# wget http://wiki.nginx.org/images/5/51/Nginx-accesskey
-2.0.3.tar.gz
```

解压：

```
[root@mail ~]# tar -zxvf Nginx-accesskey-2.0.3.tar.gz
nginx-accesskey-2.0.3/
nginx-accesskey-2.0.3/config
nginx-accesskey-2.0.3/ngx_http_accesskey_module.c
```

比较简单，只有两个文件，编辑“config”文件：

```
[root@mail nginx-accesskey-2.0.3]# vi config
USE_MD5=YES
USE_SHA1=YES
ngx addon name=ngx http accesskey module
HTTP_MODULES="$HTTP_MODULES ngx_http_accesskey_module"
NGX ADDON SRCS="$NGX ADDON SRCS
$ngx_addon_dir/ngx_http_accesskey_module.c"
```

在第 4 行找到字符串“$HTTP_ACCESSKEY_MODULE”，然后再利用字符串“ngx_http_accesskey_module”将其替代。

最后一步，那就是编译安装该模块：

```
./configure --prefix=/usr/local/nginx0.8.53/  --add-module=/root/nginx
-accesskey-2.0.3
```

> 注意在启动 Nginx 时，如果出现以下错误而不能够启动时，那肯定是没有将字符串“$HTTP_ACCESSKEY_MODULE”替换为“ngx_http_accesskey_module”，注意：替换后不再有“$”。

```
2010/12/09 12:55:12 [emerg] 27975#0: unknown directive "accesskey" in
/usr/local/nginx0.8/conf/sites-enabled/mail.t1.com:9
```

3. 指令

Accesskey 模块提供了以下四条指令。

指令名称：access key

语法：accesskey [on|off]

默认值：accesskey off

使用环境：main，server，location

功能：启用 access-key 限制功能

指令名称：accesskey_arg

语法：accesskey_arg“string”

默认值：accesskey“key”

使用环境：main，server，location

功能：URL 参数包含的访问 key。

指令名称：accesskey_hashmethod

语法：accesskey_hashmethod [md5|sha1]

默认值：accesskey_hashmethod md5

使用环境：main，server，location

功能：创建 key 使用 MD5 还是 SHA-1。

指令名称：accesskey_signature

语法：accesskey_signature“string”

默认值：accesskey_signature“$remote_addr”

使用环境：main，server，location

功能：将该字符串生成哈希值以便建立访问 key。默认产生的 key，包括客户端唯一的 IP 地址，为了更有把握，也可包含一个密码短语（asecret phrase），将它集成在 key 中（例如"myPassWord$remote_addr"）。

4. 使用实例

我们在 Nginx 配置文件中添加以下配置：

```
location /download  {
accesskey   on;
accesskey_hashmethodmd5;
accesskey arg   "key";
accesskey signature "12345$remote addr";
}
```

访问测试

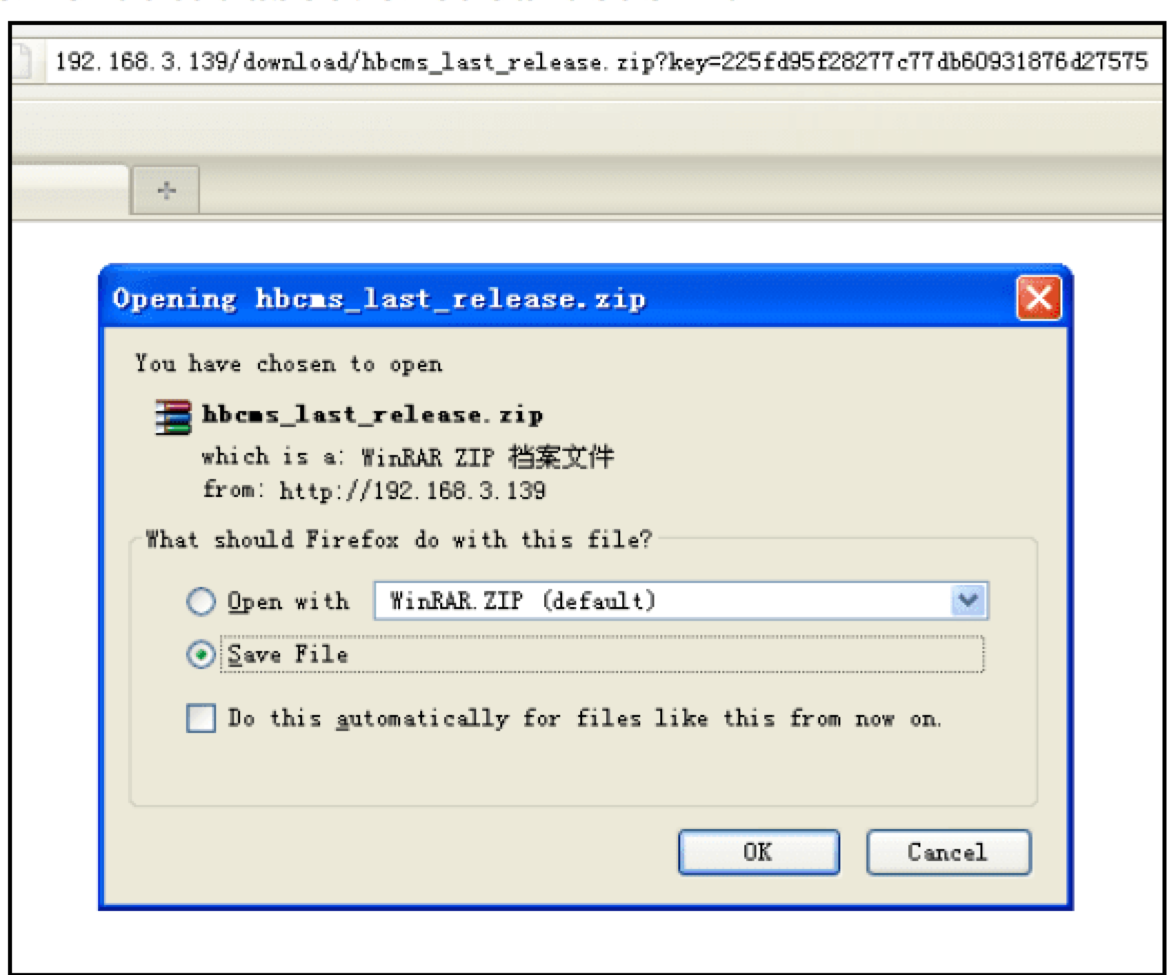
我们首先用传统的方法“http://192.168.3.139/download/hbcms_last_release.zip”下载。在浏览器中输入以下地址访问：



可见被禁止访问。

然后通过以下的方式访问：

在 http://192.168.3.139/download/hbcms_last_release.zip 的后面紧跟上字符串："?key=225fd95f28277c77db60931876d27575"。



没问题，可以下载。怎么回事呢？在正常的下载地址后添加了什么东西呢？在"http://192.168.3.139/download/hbcms_last_release.zip?key=225fd95f28277c77db60931876d27575"尾部的这个字符串就是一个验证的字符串。

它是这么来的，我们看一下服务器端的设置：

```
location /download  {
accesskey on;
accesskey hashmethod  md5;
accesskey_arg "key";
accesskey_signature"12345$remote_addr";
}
```

在上面的设置中，我们使用的哈希方法为 md5，URL 参数包含的访问 key 为"key"，即指令 accesskey_arg 的默认值，生成签名的字符串为"12345$remote_addr"，为了测试，我们使用一个 PHP 的 md5（）函数来生成该散列值：

```
[root@mail ~]# vi get.php
<?php
```

```
$secret = "12345";
$m = md5 ($secret."192.168.3.248") ;
echo $m;
echo "\n";
?>
```

执行该脚本：

```
[root@mail ~]# php  get.php

225fd95f28277c77db60931876d27575
[root@mail ~]#
```

这便是我们 key 之后的散列值。

我们在浏览器中输入散列值的方法仅仅是为了证实一下它的有效性——可以防止盗链，而事实上并不会这么去访问。真正的访问是由程序来访问的，如果你是 Java 环境，那么可以使用 Java 程序来访问，如果你是 PHP 环境，那么可以使用 PHP 程序来访问，这个我就不多说了，那是程序开发的事了。

31.3 使用 SecureLink 模块

SecureLink 模块用于为所需的安全性“令牌”计算和检查请求 URL。在 0.7.18 版本以上的 Nginx 中提供了该模块，该模块在 Nginx 的默认安装中没有包含在内，因此，如果想使用该模块则需要在 configure 时指定--with-http_secure_link_module 选项。对于 0.8.50 之后的版本，因添加了 secure_link_md5 指令和 secure_link_expires 变量，因此，指令 secure_link_secret 已经不赞成使用了。

1. 配置示例

示例 1

```
 location /prefix/ {
   secure_link_secret secret_word;

 # If the hash is incorrect then $secure_link has the value of the null
string.
   if  ($secure_link = "")  {
  return 403;
   }

   # This needs to be here otherwise you'll get a 404.
    rewrite ^ /prefix/$secure_link break;
 }
```

示例 2

```
location /p/ {
## This must match the URI part related to the MD5 hash and expiration time.
```

```
    secure_link $arg_st,$arg_e; # this must match the URI part related

    ## This is how the MD5 hash is built from a secret token, an URI and an
    ## expiration time.
    secure_link_md5 segredo$uri$arg_e; # 'segredo' is the secret token

    ## If the hash is incorrect then $secure_link is a null string.
    if ($secure_link = "") {
    return 403;
    }

    ## The current local time is greater than the specified expiration time.
    if ($secure_link = "0") {
    return 403;
    }

    ## If everything is ok $secure_link is 1.
    ## This needs to be here otherwise you'll get a 404.
    rewrite ^/p/(.*)$ /p/$1 break;
    }
```

在这个配置中，我们最终会通过以下 URL 访问：

```
    http://example.com/p/files/top_secret.pdf?st=PIrEk4JX5gJPTGmvqJG41g&e=1
324527723
```

在这个 URL 中有两处看起来不自然：

一处是 **st=PIrEk4JX5gJPTGmvqJG41g**，另一处是 **e=1324527723**，这两个参数会被传递到服务器端，然后通过相应的参数会获取这些值。对于这些值的使用，后面会讲到，我们现在主要分析这些值是怎么来的。下面的值都是通过在命令行中计算出的，如果在具体的应用中则都是通过具体的语言自动计算得出。

要构建上面的哈希值，可以使用 PHP 语言（当然其他语言也是可以的），例如下面是在命令行中使用 PHP 生成哈希值：

```
    [root@mail gz]# php -r 'print str_replace("=", "",strtr(base64_encode
(md5("segredo/p/files/top_secret.pdf1324527723", TRUE)), "+/", "-_")) .
"\n";'
    PIrEk4JX5gJPTGmvqJG41g
```

如上所示，加粗的一行便是 MD5 哈希值。当然如果你运行着 Web 应用，那么这个值必须靠自动生成，而不能像这样通过手动命令行来操作。

> 需要注意的是，MD5 哈希格式为二进制格式，因此要进行 Base64 编码。

对于生存期，我们可以通过 PHP 的 time() 函数来实现，当然也可使用其他语言来实现，为了获取 Unix epoch 时间格式，在这里我们可以通过 Linux 命令计算出：

```
[root@mail gz]# date +%s -d "December 22, 2011  12:22:03"
1324527723
```

也许你会问，这个“December 22，2011　12:22:03”时间是如何推算出来的，同样是使用 date：

```
[root@mail gz]# date -d @1324527723
Thu Dec 22 12:22:03 CST 2011
```

2. 指令

Secure Link 模块提供了以下 3 条指令。

指令名称：secure_link_secret

语法：secure_link_secret secret_word

默认值：none

使用环境：location

功能：该指令用于指定一个密码，该密码被用于 MD5 哈希生成校验请求。一个完整的被保护的连接格式：/prefix/MD5 hash/reference。

这里的 MD5 哈希值就是由该指令指定的 secret_word 密码生成，然后利用它来保护安全连接 URI。

例如，我们想保护位于目录 p 下的文件 top_secret_file.pdf，那么我们需要在 Nginx 的配置文件中添加以下配置：

```
location /p/ {
secure link secret segredo;

# If the hash is incorrect then $secure_link has the value of the null string.
if ($secure_link = "") {
return 403;
}

# This needs to be here otherwise you'll get a 404.
rewrite ^ /p/$secure link break;
}
```

我们可以通过使用 openssl 目录行工具来计算 MD5 哈希值，具体的做法是这样的：

```
[root@mail gz]# echo -n 'top secret file.pdfsegredo' | openssl dgst -md5
0849e9c72988f118896724a0502b92a8
```

我们看到，被 MD5 计算的字符不仅仅是指令 secure_link_secret 指定的 segredo 密码，还有被访问文件的文件名称。

在计算出这个值后，才可以使用以下的 URL 进行访问（这已经是一个被保护的 URL）：

```
http://example.com/p/0849e9c72988f118896724a0502b92a8/top_secret_file.pdf
```

而采用通常的方法：

```
http://example.com/p/top_secret_file.pdf
```

是无法访问到文件的。

需要注意的还有一点，那就是不要出现以下的使用方法：

```
location / {
   # This is wrong, wrong, wrong. It's a root path!
   secure_link_secret segredo;
   […]
}
```

该配置之所以错，就是因为它对根路径实施了保护，这是不可以的。因此，仅能对非根的路径进行安全连接保护。

指令名称：secure_link

语法：secure_link md5_hash[,expiration_time]

默认值：none

使用环境：location

功能：该指令指定了 MD5 哈希值和连接 URI 的生存期时间。这个 md5_hash 应该使用 Base64 进行编码；expiration_time 是 UNIX epoch 格式的时间（1 小时为 3600 秒，一个基准日（也称纪元日，epoch day）是 86400 秒，闰秒没有计算在内。多数 UNIX 系统将时间戳以一个 32 位整型数进行保存，这可能会在 2038 年 1 月 19 日产生一些问题（Y2038 问题））。如果没有指定 expiration_time，那么连接永不过期。

指令名称：secure_link_md5

语法：secure_link_md5 secret_token_concatenated_with_protected_uri

默认值：none

使用环境：location

功能：该指令指定了一个被 MD5 哈希计算的字符串，字符串中可以使用变量，计算所得出的哈希值会与指令 secure_link 中给定的 md5_hash 值进行比较，如果它们一致，那么变量$secure_link 的值将会等于 1，否则，将会是一个空字符串。

3. 变量

SecureLink 模块提供了以下两个变量。

变量名称：$secure_link

功能：该变量的值依赖于是否使用 secure_link_secret 指令，如果使用该指令，那么当请求的 URI 正确，例如，匹配 MD5 哈希，那么$secure_link 等于被保护的 URI，否则为空字符串；如果使用了 secure_link 指令和 secure_link_md5 指令，当请求的 URI 正确，例如，匹配 MD5 哈希，那么$secure_link 等于 1。如果当前的本地时间超过$expiration_time 时间，那么$secure_link 为 0，否则为空字符串。

变量名称：$secure_link_expires

功能：如果指定了生存期，那么该变量的值等于$expiration_time。

4. 实用实例

在这里我们举两个例子，一个是针对 secure_link_md5 指令的使用，另一个是针对不推荐使用的指令 secure_link_secret，虽然这种方法已经不赞成使用了，但是根据我们具体的使用环境还

是可以使用的，简单有简单的好处。

实例 1

在 Nginx 中添加以下配置内容：

```
server {
  listen   80;
  server name  www.xx.com;

  location / {
root   html;
index  index.html index.htm;
  }
location /pic/ {
  secure link $arg st,$arg e;
  secure link md5 lzbzd@123$uri$arg e;

  if  ($secure_link = "")  {
  return 403;
  }

  if  ($secure_link = "0")  {
  return 403;
  }

  rewrite ^/pic/ (.*) $ /pic/$1 break;
  }
```

下面我们通过以下 URL 访问：

```
http://www.xx.com/pic/w2-2-7-4-1.jpg
```

浏览器显示的当然是“403 Forbidden”了，这里就不再截图了。下面看一下 Nginx 的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.248 - - [31/Aug/2011:11:01:27 +0800] "GET /pic/w2-2-7-4-1.jpg
HTTP/1.1" 403 168 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

下面我们再通过以下 URL 访问：

```
http://www.xx.com/pic/w2-2-7-4-1.jpg?st=0qoAC gmNk3uDv8Fb4ZASw&e=13561501
23
```

生成哈希值：

```
[root@mail gz]# php -r 'print  str replace("=", "",strtr(base64 encode
(md5("lzbzd@123/pic/w2-2-7-4-1.jpg1356150123", TRUE)), "+/", "-_")) . "\n";'
0qoAC_gmNk3uDv8Fb4ZASw
```

生成有效的生存期。

```
[root@mail gz]# date +%s -d " December 22, 2012  12:22:03"
1356150123
```

看一下访问日志

```
192.168.3.248 - - [31/Aug/2011:11:01:40 +0800] "GET /pic/w2-2-7-4-1.jpg?st
=0qoAC_gmNk3uDv8Fb4ZASw&e=1356150123 HTTP/1.1" 200 29571 "-" "Mozilla/5.0
(Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12
GTB7.1"
```

访问响应代码为 200，可见访问成功。

在这个例子中我们使用手动的方式生成了哈希值和生存期。同样我们可以将这些工作封装在 PHP 或者是其他语言构成的程序中来自动完成。这些工作就不是运维所做的了，那是开发人员的事情了（可千万别和我说你还兼做开发人员！）。

实例 2

在 Nginx 的配置文件中添加以下配置内容：

```
server {
  listen   80;
  server_name  www.xx.com;

  location / {
root   html;
index  index.html index.htm;
  }

location /pic/ {
 secure_link_secret lzbzd@123;

 if  ($secure_link = "")  {
   return 403;
   }

    rewrite ^ /pic/$secure_link break;
 }
```

我们通过以下 URL 地址访问：

```
http://www.xx.com/pic/w2-2-7-4-1.jpg
```

浏览器显示的当然是“403 Forbidden”了，这里就不再截图了。下面看一下 Nginx 的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.248 - - [31/Aug/2011:13:02:06 +0800] "GET /pic/w2-2-7-4-1.jpg
HTTP/1.1" 403 570 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
```

我们再通过以下 URL 地址访问：

```
http://www.xx.com/pic/9c611322a0795492c09b5f4c9a02c3b6/w2-2-7-4-1.jpg
```

首先生成 MD5 哈希值，然后再访问：

```
[root@mail gz]# echo -n 'w2-2-7-4-1.jpglzbzd@123' | openssl dgst -md5
9c611322a0795492c09b5f4c9a02c3b6
```

这是我们的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.248 - - [31/Aug/2011:13:04:11 +0800] "GET /pic/9c611322a0795492
c09b5f4c9a02c3b6/w2-2-7-4-1.jpg  HTTP/1.1"  200  29571  "-"  "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
```

可以看到访问响应代码为 200，即访问成功。

第 32 章　Nginx 提供 HTTPS 服务

在 Nginx 服务器中要想提供 HTTPS 服务，那么需要安装 SSL 模块。下面的篇幅中我将会了解到该模块。

32.1　兼容性

- 从 0.8.21 和 0.7.62 版本开始，通过“-V”参数可以显示出 SNI 状态；
- 从 0.7.14 版本后，“listen”指令开始支持“ssl”；
- 从 0.5.32 版本后，支持 SNI；
- 从 0.5.6 版本后，开始支持共享 SSL 会话缓存；
- 版本 0.7.65，0.8.19 及以上：默认的 SSL 协议是 SSLv3 和 TLSv1；
- 版本 0.7.64，0.8.18 及以上：默认的 SSL 协议是 SSLv2，SSLv3，和 TLSv1；
- 版本 0.7.65，0.8.20 及以上：默认的 SSL 密码是“HIGH:!ADH:!MD5”；
- 版本 0.8.19：默认的 SSL 密码是“ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM”；
- 版本 0.7.64，0.8.18 及以上：默认的 SSL 密码是“ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP”。

32.2　安装 SSL 服务

SSL 模块在默认情况下不会被包含，因此，如果要使用该模块，那么在编译安装 Nginx 时必须明确指定--with-http_ssl_module 参数。另外，该模块需要 OpenSSL 库和相关的开发包，即 libssl-dev，或者是类似的包。

```
[root@mail nginx-1.0.2]# ./configure --prefix=/usr/local/nginx-1.0.2-ssl
--with-http_ssl_module
```

要配置一个 HTTPS 服务器，必须在 server 区段中开启 SSL 协议，并且需要指定服务器证书和私钥文件的位置：

```
server {
listen 443;
server_name  www.nginx.com;
ssl  on;
ssl_certificate  www.nginx.com.crt;
ssl_certificate_key  www.nginx.com.key;
ssl_protocolsSSLv3 TLSv1;
ssl_ciphers  HIGH:!ADH:!MD5;
…
```

```
}
```

服务器证书是一个公共实体（public entity），它将会被发送到每一个连接到服务器的客户端；私钥是一个安全实体（ecure entity），并将该文件存储在一个受限访问的文件中，但它必须能够被 Nginx 的 master 进程可读。私钥可以与证书交替地存储在同一个文件中：

```
ssl_certificate  www.nginx.com.cert;
ssl_certificate_key  www.nginx.com.cert;
```

在这种情况下，该文件访问权限也应该受到约束，尽管私钥与证书存储在同一个文件，但是仅有证书被发送到客户端。

指令"ssl_protocols"和"ssl_ciphers"可以设定限制用于连接 SSL 协议的版本及强壮性的密码。自从 Nginx 0.8.20 版本以来，默认使用"ssl_protocols SSLv3 TLSv1"和"ssl_ciphers HIGH:!ADH:!MD5"，因此，它们仅需要在 Nginx 早期的版本中设置。

在这个例子中，为了减少 CPU 负载，将其指到一个 worker 进程，并且启用了 keepalive。

当使用证书链文件时，只需要将另外的证书追加到.crt 文件（在上例中是 cret.pem），将服务器证书放置在文件的开始部分，否则公钥和私钥无法进行匹配。

另外，自从 0.7.14 版本之后，启用 SSL 首选连接的方法要在 listen 指令中添加 ssl 参数，例如：

0.7.14 版本以后通常将 ssl 参数使用到 listen 指令中：

```
server {
  listen 443 default server ssl;
  ssl_certificate  /usr/local/nginx/conf/cert.pem;
  ssl_certificate_key  /usr/local/nginx/conf/cert.key;
  …
```

1. 配置示例

```
worker_processes 1;
http {
  server {
listen   443;
ssl  on;
ssl certificate  /usr/local/nginx/conf/cert.pem;
ssl certificate key  /usr/local/nginx/conf/cert.key;
keepalive_timeout70;
  }
}
```

2. 指令

SSL 模块提供的指令较多，有以下 14 条指令。

指令名称：ssl

语法：ssl [on|off]

默认值：ssl off

使用环境：http，server

功能：启用 HTTPS 服务器。

指令名称：ssl_certificate

语法：ssl_certificate file

默认值：ssl_certificate cert.pem

使用环境：http，server

功能：该指令用于为虚拟主机指定包含证书的文件，使用 PEM 格式，这个文件也可以包含其他证书和服务器的私钥。从 0.6.7 版本后，指定该文件的路径要依赖于 Nginx 的配置文件 nginx.conf 所在的路径。

指令名称：ssl_certificate_key

语法：ssl_certificate_key file

默认值：ssl_certificate_key cert.pem

使用环境：http，server

功能：该指令用于为虚拟主机指定包含私钥的文件，使用 PEM 格式。从 0.6.7 版本后，指定该文件的路径要依赖于 Nginx 的配置文件 nginx.conf 所在的路径。

指令名称：ssl_client_certificate

语法：ssl_client_certificate file

默认值：none

使用环境：http，server

功能：该指令用于指定了包含 CA（根）证书的文件，使用 PEM 格式。它的功能在于确认客户端证书。

指令名称：ssl_dhparam

语法：ssl_dhparam file

默认值：none

使用环境：http，server

功能：该指令用于指定一个包含 Diffie-Hellman 密钥协议的加密参数，使用 PEM 格式，用于服务器和客户端的会话密钥交换。

指令名称：ssl_ciphers

语法：ssl_ciphers openssl_cipherlist_spec

例如：ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

默认值：ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

使用环境：http，server

功能：为建立安全连接，该指令用于指定一套服务器支持的加密方式，使用 OpenSSL 格式。如果想查看 Nginx 所在平台支持的 OpenSSL，那么可以执行 openssl ciphers，例如：

```
[root@mail conf]# openssl ciphers
   DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA:AES256-SHA:EDH-RSA-DES-CBC3-SHA:E
DH-DSS-DES-CBC3-SHA:DES-CBC3-SHA:DES-CBC3-MD5:DHE-RSA-AES128-SHA:DHE-DSS-A
ES128-SHA:AES128-SHA:RC2-CBC-MD5:DHE-DSS-RC4-SHA:EXP-KRB5-RC4-MD5:EXP-KRB5
```

```
-RC4-SHA:KRB5-RC4-MD5:KRB5-RC4-SHA:RC4-SHA:RC4-MD5:RC4-MD5:KRB5-DES-CBC3-M
D5:KRB5-DES-CBC3-SHA:EXP1024-DHE-DSS-DES-CBC-SHA:EXP1024-DES-CBC-SHA:KRB5-
DES-CBC-MD5:KRB5-DES-CBC-SHA:EDH-RSA-DES-CBC-SHA:EDH-DSS-DES-CBC-SHA:DES-C
BC-SHA:DES-CBC-MD5:EXP1024-DHE-DSS-RC4-SHA:EXP1024-RC4-SHA:EXP-KRB5-RC2-CB
C-MD5:EXP-KRB5-DES-CBC-MD5:EXP-KRB5-RC2-CBC-SHA:EXP-KRB5-DES-CBC-SHA:EXP-E
DH-RSA-DES-CBC-SHA:EXP-EDH-DSS-DES-CBC-SHA:EXP-DES-CBC-SHA:EXP-RC2-CBC-MD5
:EXP-RC2-CBC-MD5:EXP-RC4-MD5:EXP-RC4-MD5
```

指令名称：ssl_crl

语法：ssl_crl file

默认值：none

使用环境：http，server

功能：该指令在 Nginx 的 0.8.7 版本中开始使用，也就是说只有该版本之后的 Nginx 才可以使用该指令。其功能在于指定一个吊销证书列表的文件名，同样使用 PEM 格式，它可以被用于检查吊销证书状态。

指令名称：ssl_prefer_server_ciphers

语法：ssl_prefer_server_ciphers [on|off]

默认值：ssl_prefer_server_ciphers off

使用环境：http，server

功能：对于依赖于 SSLv3 和 TLSv1 协议的服务器密码，将会优先于客户端密码。

指令名称：ssl_protocols

语法：ssl_protocols [SSLv2] [SSLv3] [TLSv1]

默认值：ssl_protocols SSLv2 SSLv3 TLSv1

使用环境：http，server

功能：该指令用于启用指定的协议版本。

指令名称：ssl_verify_client

语法：ssl_verify_client on|off|optional

默认值：ssl_verify_client off

使用环境：http，server

功能：该指令用于设置启用客户端验证。参数 optional 的功能在于指出即使服务器端有效仍使用客户端自己的证书来验证。在以前的版本 0.8.7 和 0.7.63 中使用的是 ask 参数。

指令名称：ssl_verify_depth

语法：ssl_verify_depth number

默认值：ssl_verify_depth 1

使用环境：http，server

功能：该指令用于设置服务器按序检查客户端提供的证书链的长度。

指令名称：ssl_session_cache

语法：ssl_session_cache off|none|builtin:size 和/或 shared:name:size

默认值：ssl_session_cache off

使用环境：http，server

功能：该指令用于设置缓存的类型和大小，用于存储 SSL 会话。

- off，硬性关闭（或者叫强制关闭）：Nginx 服务器明确指出客户端会话不能够被重新使用。
- none，软关闭（或者叫非强制关闭）：Nginx 服务器指出客户端会话能够被重新使用，但实际上 Nginx 并不会重新使用它们。这么做的原因在于有些邮件客户端会使用指令 ssl_session_cache，也就是说只是为了这些客户端的使用才使用。
- builtin：使用 OpenSSL 内建的缓存，仅能在一个 worker 处理进程中。缓存的大小在会话中指定。注意：使用这种方法会出现内存碎片问题，因此要谨慎使用这种方式。
- shared：使用这种方式，所有的 worker 进程将会共享这个缓存，指定缓存大小的单位为字节：1MB 的缓存能够容纳 4000 个会话。每一个共享缓存必须指定一个自己的名称（名称要唯一），这个共享的缓存可以用在多个虚拟主机中。可以同时使用两种类型的缓存——内置和共享，例如：

```
ssl_session_cache  builtin:1000  shared:SSL:10m;
```

> 然而，要注意的是，仅使用共享缓存类型。例如，在这里没有使用内置类型，那么会更高效。

对于 0.8.34 以下版本的 Nginx，如果将 ssl_verify_client 设置为“on”或“optional”，那么该指令不能够设置为“none”或“off”。

> 注意，为了能够使得会话继续工作，至少需要为 SSL 设定一个 **default** 。例如：“listen [::]:443 ssl default_server”。

指令名称：ssl_session_timeout

语法：ssl_session_timeout time

默认值：ssl_session_timeout 5m

使用环境：http，server

功能：该指令用于设定客户端在安全会话中能够重新使用先前自动协商加密参数（即存储在 SSL 缓存中的安全会话）的最长期限。

指令名称：ssl_engine

语法：ssl_engine engine

默认值：依赖于系统。

使用环境：http，server

功能：该指令用于指定 OpenSSL 使用的引擎，例如像 PadLock。可以通过该指令来改变引擎。使用 openssl engine 命令可以获取当前系统中的引擎，例如：

```
[root@mail conf]#  openssl engine
(dynamic) Dynamic engine loading support
(padlock) VIA PadLock (no-RNG, no-ACE)
```

32.3 通配符证书

在多个 server 中使用通配符证书。在具体的使用中，可能会在多个非安全子域中提供安全域。要想这么做，那么需要一个通配符子域，例如，*.nginx.org，只有在这种情况下才可以实现。在下面的例子中，展示了如何配置一个标准的 www 子域、一个安全的子域和一个用于前两者（即 HTTP 和 HTTPS）共享访问的图片子域。

要实现这种配置，将包含证书的文件私钥文件放置在 http 区段部分，这样每一个 server 或者是虚拟主机将都会继承这些设置，例如：

```
ssl_certificate  common.crt;
ssl_certificate_key  common.key;

server {
  listen   80;
  server_name  www.nginx.org;
  …
}

server {
  listen   443 default_server ssl;
  server_name  secure.nginx.org;
  …
}

server {
  listen   80;
  listen   443;
  server_name  images.nginx.org;
  …
}
```

32.4 变量

SSL 模块 ngx_http_ssl_module 支持下列内置变量。

- $ssl_cipher：该变量会返回被用于建立的 SSL/TLS 连接中那些使用的密码字段。
- $ssl_client_serial：该变量会返回用于当前建立 SSL/TLS 连接的客户端证书的序列号，前提条件是在连接中客户端认证有效。
- $ssl_client_s_dn：返回当前已建立的 SSL/TLS 会话中客户端证书 subject 部分的 DN，前提条件是在连接中客户端认证有效。
- $ssl_client_i_dn：返回当前已建立的 SSL/TLS 会话中客户端证书发行者的 DN，前提条件

是在连接中客户端认证有效。

- $ssl_protocol：返回当前已建立的 SSL/TLS 会话中所使用的协议，其值依赖于服务器端的配置和客户端有效的选项，可能的值有 SSLv2，SSLv3 或 TLSv1。
- $ssl_session_id：返回当前已建立的 SSL/TLS 会话中的会话 ID，它需要 0.8.20 以上的 Nginx 版本才支持。
- $ssl_client_cert：返回当前已建立的 SSL/TLS 会话中客户端的公钥，返回格式为 Base64 明文，并且将原始证书中所有的 Windows 回车替换为 UNIX 回车。
- $ssl_client_raw_cert：返回当前已建立的 SSL/TLS 会话中客户端的原始公钥。
- $ssl_client_verify：当客户端证书审核通过，那么将会返回一个“SUCCESS”值。

32.5 非标准的错误代码

SSL 模块支持一些非标准的错误代码，可以借助于 error_page 指令来调试：

- 495：检查客户端证书时发生错误；
- 496：客户不能够提供授权的证书；
- 497：传递到 HTTPS 的正常请求（就是说 HTTP 请求）。

32.6 使用举例

既可以实现 SSL 单向认证，即服务器端认证，也可以实现双向认证，即服务器和客户端的双向认证。

32.6.1 单向认证

要产生私有证书，那么可以在 Linux 系统中执行 openssl 命令，首先我们改变目录到放置证书的目录，例如：

```
[root@mail nginx-1.0.2]# cd /usr/local/nginx-1.0.2-ssl/conf
[root@mail conf]#
```

现在我们来建立服务器的私钥，在这个过程中需要输入密码短语（要记住这个密码）：

```
[root@mail conf]# openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
........++++++
.....................................................................
.................................++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

看一下该文件的内容：

```
[root@mail conf]# more server.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,C879602653AF22F9

YgoJCBl/dOnuUsyqfiEEWDfJ1yV/tot+GELb/9PAVzk4fXcGuKJhxNHLgrUA2Aqz
xRw2IZFLUeFVigeT9/5PrEDdGelAIQZ7Md47cCTbLE4EaZL1tqaTCKCvAGSvWp2+
TZgyZ1duzTXZ4hC4qnJ5vQDDl/ZF5b2Ux6zOESZsr06Fo3ndttPrKyQ0rKfkD0Jb
ojVGN9GDWyel8l8xyPwfjXqVLQS3eG5r5Y0MwtiXD4pGlY1FFv5vGnFL2fUkuxkv
AHkJwi2WhxvtvHRl3C90KXEQ1rNzc99sjIApAYhh9BbID5DIjTLSjMS0ErBFVrUF
lJqfFoHru9DQd4kIl2r0ZMcEfQv5ZQWZEvekQWiKbuI5k934mKuCDuUdjjmV9Q5v
HH0ERB30H5owINngRij7UNn3xqITwdwY30tAGYZErcq2IZghoAICPYjy2K85/2QE
gzsI0xhgrugmZr8ffJMC6Z3vEemX84SXcCk2cx2t+3OqHtAx/XvCwNVJgcTB1lh9
sisIBqsBCHqhpXbg+ATxKa4mjencnud7L9Eg7JqVpi4xcBVaBq853wEkckqaidHg
8eZTWO4FQkypEPKKxby9b2vGRSUNXBWEY+CqzFoCTg+N/6EohYxxDtmd1ABXUJHH
7+HM9zwFKtTeCbRDWkng87zfbrYvCnisMeZasCeSBL0n4nfOstIuEqqIvDh7FCjx
FZqVzrrbA4kPVxEUjel8imeHsJMYZuntG2Dy/JzdhQWQfwUBwD632jzEYkuPg3aR
OHDI+UPLtCS94O5Eb+RCRgtmCnAQpjGPYSAD+bHtuu2WkiLMG4qMlw==
-----END RSA PRIVATE KEY-----
```

创建证书签名请求（CSR）：

```
[root@mail conf]# openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:CN
State or Province Name (full name) [Berkshire]:BEIJING
Locality Name (eg, city) [Newbury]:BEIJING
Organization Name (eg, company) [My Company Ltd]:BIRD
Organizational Unit Name (eg, section) []:mc
Common Name (eg, your name or your server's hostname) []:www.xx.cn
Email Address []:ma@xx.cn

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:sdf6@fdg
An optional company name []:mc
```

看一下 server.csr 的内容：

```
[root@mail conf]# more server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIB0jCCATsCAQAwezELMAkGA1UEBhMCQ04xEDAOBgNVBAgTB0JFSUpJTkcxEDAO
BgNVBAcTB0JFSUpJTkcxDjAMBgNVBAoMBUJJUkQIMQswCQYDVQQLEwJtYzESMBAG
A1UEAxMJd3d3Lnh4LmNuMRcwFQYJKoZIhvcNAQkBFghtYUB4eC5jbjCBnzANBgkq
hkiG9w0BAQEFAAOBjQAwgYkCgYEAtSnMmuyP/haeqzfFbaxio3Xq5N+a8hZOjV+b
G1ePwG98weIPcXW10czfy6KkHI81JT3cQtqyuLyr4u/gL+R3MZfhJXzBVVPW1/w9
K4LM0yQmyoFnXd6oFDYWab9SPiFbBGBFxYqjZ0GQLNQ1gd156uI4hKqV78AQsIZ/
catS720CAwEAAaAXMBUGCSqGSIb3DQEJBzEIEwYxMjM0NTYwDQYJKoZIhvcNAQEF
BQADgYEAl6HCcjHYHerV92vvS99EeAhHKmwaq4AJ4CRZqaueGtUzZppPWl8vYEX4
kmGP2ve5ppfb72AoUnf5NO1elsfMtIzYUo51b1Xc/+2jZ8J95yNcRXm0Ow+fbIlv
XgLbpR/qeeTiwPuTOMm4kGxLTa7s/q2F2zamOMBDy3X8RWsxMpU=
-----END CERTIFICATE REQUEST-----
```

对于使用上面的私钥启动具有 SSL 功能的 Nginx，有必要移除输出的密码：

```
[root@mail conf]#  cp server.key server.key.org
[root@mail conf]# openssl  rsa -in server.key.org -out server.key
Enter pass phrase for server.key.org:
writing RSA key
```

看一下 server.key 现在的内容：

```
[root@mail conf]# more server.key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQClKcya7I/+Fp6rN8VtrGKjderk35ryFk6NX5sbV4/Ab3zB4g9x
dbXRzN/LoqQcjzUlPdxC2rK4vKvi7+Av5Hcxl+ElfMFVU9bX/D0rgszTJCbKgWdd
3qgUNhZpvlI+IVsEYEXFiqNnQZAslDWB3Xnq4jiEqpXvwBCwhn9xqlLvbQIDAQAB
AoGAcsVDYVbDNnbAGLqhcgTHDMWUhWD/uaTWKQxLqHvFj2VSBUahAAFRqdOAaHLm
Q02anrhdBWGl3hFxV7keL1kvBQQLjY62skF/a42kI+Hd8rVkJBNKxfyI4shxwWW9
/G/bfT3ff1ldgSHlJDBJDhYj0kvUzAMwG94DvyUKiLlO7eECQQDhZ8WVN+3360FF
GwR5+ARzC3i7R73A48a4LzAiLWFiG4BeU+lxoFCCC5wwSaEqmRC9e0SjwtVcFLVS
9935oUbFAkEAzcC8BtuR8HH/cSWPGHTjSjMXkl/doY53eRpVs9A/SNVYYcg+u9KB
SqRl3ylRckEttI0PC5PnG8D0Xu7ZqLTQiQJBAMvRBbxUAn323+IY+AdB2OQeL2FJ
Ea/lRr8tDB0bY79Olti0j5YLHEE7NlQDgBQArD5pxDs/5aWJpBwNvU4IIvECQG/1
5tscdAeRIN0jAQDf6iagMr116RZWffCqQeR891LctSFQY8K0w6IoZ/OteiJgEUzk
TUGRk/roPpHCCzLgYXECQQC4RwiUOOH8W+TX5jLcCd/7n/i00cOWQd+HhJgYU3VH
DXSGYLmTwQDeiCdWDqwGalWSwI1O3G8t57Wbvp36c7kR
-----END RSA PRIVATE KEY-----
```

最后，使用上面的私钥和 CSR 对证书进行签名：

```
[root@mail conf]# openssl x509 -req -days 365 -in server.csr -signkey
server.key -out server.crt
Signature ok
subject=/C=CN/ST=BEIJING/L=BEIJING/O=BIRD\x08/OU=mc/CN=www.xx.cn/emailA
ddress=ma@xx.cn
Getting Private key
```

看一下 server.crt 文件：

```
[root@mail conf]# more server.crt
-----BEGIN CERTIFICATE-----
MIICbTCCAdYCCQCr+iQyThMhSDANBgkqhkiG9w0BAQUFADB7MQswCQYDVQQGEwJD
TjEQMA4GA1UECBMHQkVJSklORzEQMA4GA1UEBxMHQkVJSklORzEOMAwGA1UECgwF
QklSRAgxCzAJBgNVBAsTAm1jMRIwEAYDVQQDEwl3d3cueHguY24xFzAVBgkqhkiG
9w0BCQEWCGlhQHh4LmNuMB4XDTExMDgwODAyMTQzOFoXDTEyMDgwNzAyMTQzOFow
ezELMAkGA1UEBhMCQ04xEDAOBgNVBAgTB0JFSUpJTkcxEDAOBgNVBAcTB0JFSUpJ
TkcxDjAMBgNVBAoMBUJJUkQIMQswCQYDVQQLEwJtYzESMBAGA1UEAxMJd3d3Lnh4
LmNuMRcwFQYJKoZIhvcNAQkBFghtYUB4eC5jbjCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAtSnMmuyP/haeqzfFbaxio3Xq5N+a8hZOjV+bGlePwG98weIPcXW1
0czfy6KkHI81JT3cQtqyuLyr4u/gL+R3MZfhJXzBVVPW1/w9K4LM0yQmyoFnXd6o
FDYWab9SPiFbBGBFxYqjZ0GQLNQ1gdl56uI4hKqV78AQsIZ/catS720CAwEAATAN
BgkqhkiG9w0BAQUFAAOBgQCxDfR99gdyKaMU8/TfJFL23JbK8eeanwMc+eIbbAUY
DroV2JbmFjxgE/+SpKxD6uUTFLEL/PEPke46/t3Sgk2KS6D7+laBoYM0+y6ygGT4
sfdbY7OXnLvugeAh3qLk64lIBqARuUQuxxLeqxlHHZBptMiRPZLYjUAkFkzcWnr3
aw==
-----END CERTIFICATE-----
```

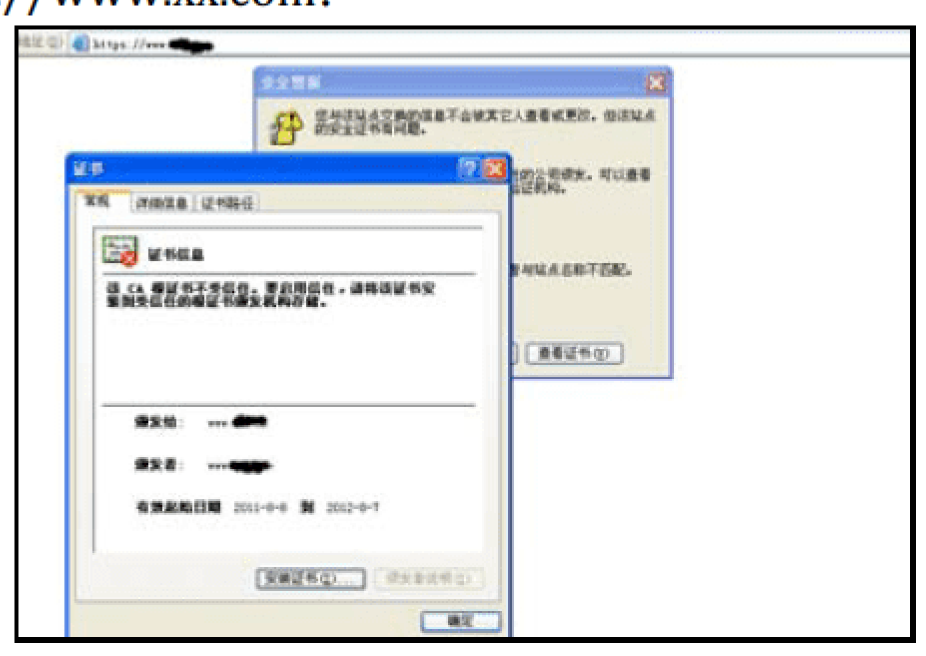
32.6.2 更新 Nginx 配置

在 Nginx 的配置文件中，添加或者修改相关选项如下：

```
server {
server_name  www.xx.com;
listen 443;
ssl on;
ssl certificate  /usr/local/nginx-1.0.2-ssl/conf/server.crt;
ssl certificate key  /usr/local/nginx-1.0.2-ssl/conf/server.key;
}
```

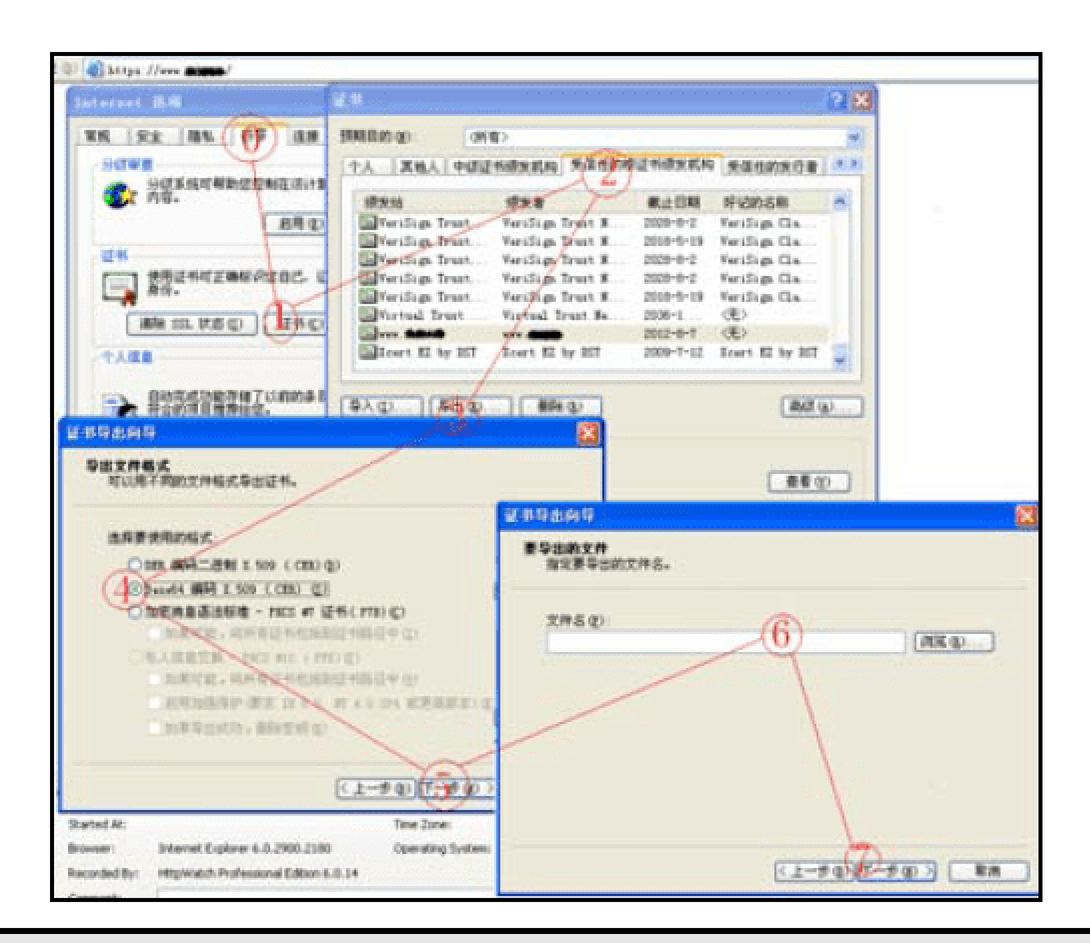
32.6.3 访问测试

启动 Nginx 访问 http://www.xx.com：



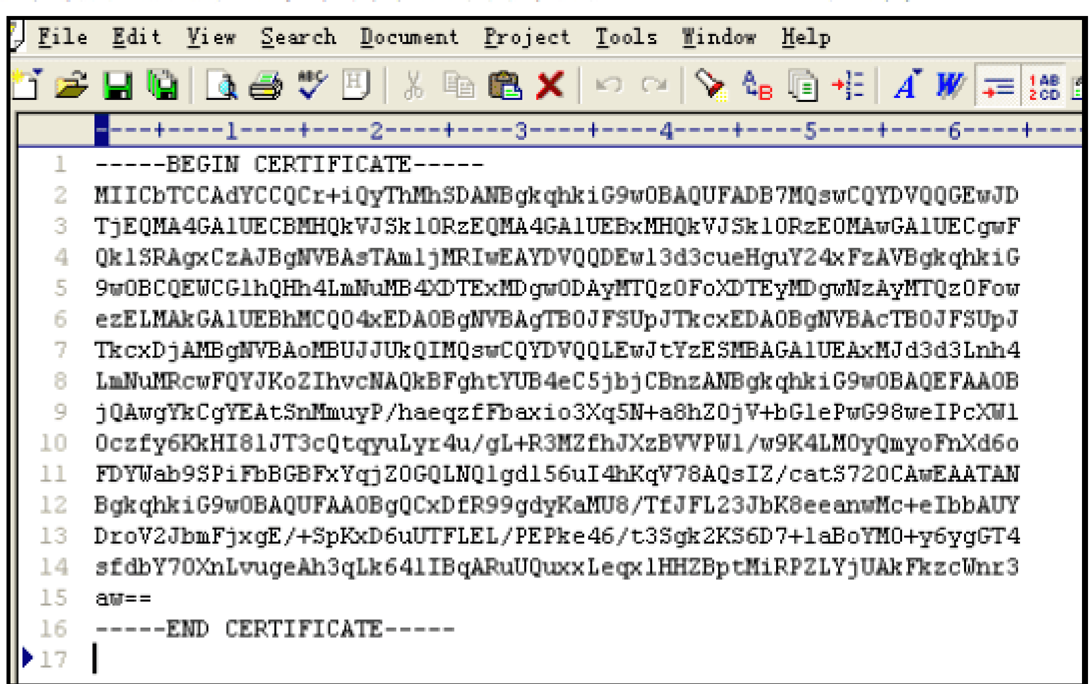
安装证书就可以了。

我们看一下证书，首先导出证书：



在上面的步骤中，在第 4 步时要选择“Base64 编码 X.509 （.CER） （S）”这一项。

查看证书。下面是我们通过文本编辑器打开的 server.crt 文件：




```
-----BEGIN CERTIFICATE-----
MIICbTCCAdYCCQCr+iQyThMhSDANBgkqhkiG9w0BAQUFADB7MQswCQYDVQQGEwJD
TjEQMA4GA1UECBMHQkVJSkl0RzEQMA4GA1UEBxMHQkVJSkl0RzEOMAwGA1UECgwF
Qk1SRAgxCzAJBgNVBAsTAmljMRIwEAYDVQQDEwl3d3cueHguY24xFzAVBgkqhkiG
9w0BCQEWCGlhQHh4LmNuMB4XDTExMDgw0DAyMTQz0FoXDTEyMDgwNzAyMTQz0Fow
ezELMAkGA1UEBhMCQ04xEDA0BgNVBAgTB0JFSUpJTkcxEDA0BgNVBAcTB0JFSUpJ
TkcxDjAMBgNVBAoMBUJJUkQIMQswCQYDVQQLEwJtYzESMBAGA1UEAxMJd3d3Lnh4
LmNuMRcwFQYJKoZIhvcNAQkBFghtYUB4eC5jbjCBnzANBgkqhkiG9w0BAQEFAA0B
jQAwgYkCgYEAtSnMmuyP/haeqzfFbaxio3Xq5N+a8hZ0jV+bGlePwG98weIPcXWl
0czfy6KkHI8lJT3cQtqyuLyr4u/gL+R3MZfhJXzBVVPWl/w9K4LM0yQmyoFnXd6o
FDYWab9SPiFbBGBFxYqjZ0GQLNQlgdl56uI4hKqV78AQsIZ/catS720CAwEAATAN
BgkqhkiG9w0BAQUFAA0BgQCxDfR99gdyKaMU8/TfJFL23JbK8eeanwMc+eIbbAUY
DroV2JbmFjxgE/+SpKxD6uUTFLEL/PEPke46/t3Sgk2KS6D7+laBoYM0+y6ygGT4
sfdbY70XnLvugeAh3qLk64lIBqARuUQuxxLeqxlHHZBptMiRPZLYjUAkFkzcWnr3
aw==
-----END CERTIFICATE-----
```


32.6.4　双向认证

在下面的例子中我们详细了解一下证书，包括 CA。关于 OpenSSL 的安装就不再讲述了，但是它的配置文件不得不说。

我们的 OpenSSL 安装在以下位置：

```
[root@mail ~]# tree -L 2  /usr/local/openssl/
/usr/local/openssl/
|-- bin
|   |-- c_rehash
|   '-- openssl
|-- include
|   '-- openssl
|-- lib
|   |-- engines
|   |-- libcrypto.a
|   |-- libssl.a
|   '-- pkgconfig
```

```
'-- ssl
|-- certs
|-- man
|-- misc
|-- openssl.cnf
'-- private
```

配置文件 openssl.cnf 的内容较多，在此就不全部提供了，我们根据需要来进行讲述。因为我们首先要设置 CA，因此，看这一部分内容：

```
####################################################################
[ ca ]
default_ca  = CA_default# 默认 CA 的配置

####################################################################
[ CA default ]

dir = ./demoCA                      # 与 CA 相关的证书
certs   = $dir/certs                # 设定被颁发的证书保存的位置
crl_dir = $dir/crl                  # 设定证书吊销列表（CRL）
database= $dir/index.txt            # 数据库 index 文件
#unique_subject = no                # 同一个 subject 是否只能创建一个证书
                                    # 设为 no 表示可以创建多个
new_certs_dir   = $dir/newcerts # 设定默认放置新证书的位置

certificate = $dir/cacert.pem       # 设定 CA 证书的位置
serial  = $dir/serial               # 当前的序列号
crlnumber   = $dir/crlnumber        # 设定当前的 crl 号，即存放当前 CRL 编号的
                                    # 文件，对于 v1 版本的 CRL 则必须注释掉该行

crl = $dir/crl.pem                   # 当前的 CRL 文件
private_key = $dir/private/cakey.pem  # 设定私钥
RANDFILE= $dir/private/.rand          # 私有的随机数文件

…                                     #省略部分

                                      # 设定 CA 策略
[ policy match ]
countryName = match
stateOrProvinceName = match
organizationName= match
organizationalUnitName  = optional
commonName  = supplied
emailAddress= optional
```

按照配置文件的要求，我们需要创建 demoCA、certs、crl、newcerts 和 private 目录，以及

crlnumber 和 index.txt 文件，总之我们可以按照这个配置文件来修改将要创建的证书及目录，也可以根据需要来修改这个配置文件。实际上只修改配置文件会更简单，但是我们为了了解一下 OpenSSL 的配置文件，因此就麻烦了一点。

32.6.5 创建相关目录

```
[root@mail ssl]# pwd
/usr/local/openssl/ssl
[root@mail ssl]# mkdir demoCA
[root@mail ssl]# cd demoCA/
[root@mail demoCA]# mkdir certs
[root@mail demoCA]# mkdir crl
[root@mail demoCA]# mkdir private
[root@mail demoCA]# mkdir newcerts
[root@mail demoCA]# touch index.txt
[root@mail demoCA]# touch crlnumber
[root@mail demoCA]# touch serial
```

1. 生成 CA

生成 CA 私钥

由于我们是自己签发证书，因此需要生成一个 CA。

```
[root@mail ssl]# /usr/local/openssl/bin/openssl genrsa -out private/ca.key
Generating RSA private key, 512 bit long modulus
.++++++++++++
.++++++++++++
e is 65537 (0x10001)
[root@mail ssl]# cd private/
[root@mail private]# ls
ca.key
[root@mail private]# more  ca.key
-----BEGIN RSA PRIVATE KEY-----
MIIBOQIBAAJBAKgOibG5uYj2r/X8ufd2ghybJch/wn3f2xJUzRHK1C543qSLHopG
hZWm15Qk6U54Z23gQhlLSpeeSVf+DvverAMCAwEAAQJAN/NXHmeCALp4jMIOO/gl
i1aP9reqTfQIYIsBFypbB/lFkxa79FNzOmt8w36v710k9TOciET6GW7neq1LsahN
iQIhANhB9BwZ0ccJPWi3LU8Gxsq8vI3Dq/kl09r9i11WJExFAiEAxvDuAigwOr4g
oNhx2rds2oItgSgvxOoxk3ImrxQTb6cCIF7JQaqTcowPs7fTGevaZ4VzBh4I1rbE
asBAgzYszIoVAiBTMHqGgmGwnsKsH/Z0PFGTie4XXUOkdz415w0QFzgNbwIgUKPc
nWXDlhxrQqg1bvi8JsaWAzIMmC2jyxKDLJWNO8w=
-----END RSA PRIVATE KEY-----
```

生成请求证书

```
[root@mail private]# cd ..
[root@mail    ssl]#    /usr/local/openssl/bin/openssl    req    -new    -key
private/ca.key -out private/ca.csr
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:bj
Locality Name (eg, city) []:beijing
Organization Name (eg, company) [Internet Widgits Pty Ltd]:pb
Organizational Unit Name (eg, section) []:pba
Common Name (eg, YOUR name) []:ym
Email Address []:ms@xx.cn

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:dd
[root@mail ssl]# cd private/
[root@mail private]# ls
ca.csr  ca.key
[root@mail private]# more ca.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBUTCB/AIBADBtMQswCQYDVQQGEwJDTjELMAkGA1UECBMCYmoxEDAOBgNVBAcT
B2JlaWppbmcxCzAJBgNVBAoTAnBiMQwwCgYDVQQLEwNwYmExCzAJBgNVBAMTAnlt
MRcwFQYJKoZIhvcNAQkBFghtc0B4eC5jbjBcMA0GCSqGSIb3DQEBAQUAA0sAMEgC
QQCoDomxubmI9q/1/Ln3doIcmyXIf8J939sSVM0RytQueN6kix6KRoWVpteUJOlO
eGdt4EIZS0qXnklX/g773qwDAgMBAAGgKjARBgkqhkiG9w0BCQIxBBMCZGQwFQYJ
KoZIhvcNAQkHMQgTBjEyMzQ1NjANBgkqhkiG9w0BAQUFAANBAAqBWHNxu67fYuYe
Ua39UC2Ok0H+Les18HKlJxtfvZS5gjprg3XzX8CVNvYAqg70qJUh5K38Fh7/cG5V
3B+CCkc=
-----END CERTIFICATE REQUEST-----
```

签名

```
[root@mail private]# cd ..
[root@mail ssl]# /usr/local/openssl/bin/openssl x509 -req -days 365 -in
private/ca.csr -signkey private/ca.key -out private/ca.crt  Signature ok
subject=/C=CN/ST=bj/L=beijing/O=pb/OU=pba/CN=ym/emailAddress=ms@xx.cn
Getting Private key
[root@mail ssl]# cd private/
[root@mail private]# more ca.crt
-----BEGIN CERTIFICATE-----
```

```
MIIBzDCCAXYCCQCOLz2DtTSHIDANBgkqhkiG9w0BAQUFADBtMQswCQYDVQQGEwJD
TjELMAkGA1UECBMCYmoxEDAOBgNVBAcTB2JlaWppbmcxCzAJBgNVBAoTAnBiMQww
CgYDVQQLEwNwYmExCzAJBgNVBAMTAnltMRcwFQYJKoZIhvcNAQkBFghtc0B4eC5j
bjAeFw0xMTA4MDgxMDUxNTVaFw0xMjA4MDcxMDUxNTVaMG0xCzAJBgNVBAYTAkNO
MQswCQYDVQQIEwJiajEQMA4GA1UEBxMHYmVpamluZzELMAkGA1UEChMCcGIxDDAK
BgNVBAsTA3BiYTELMAkGA1UEAxMCeW0xFzAVBgkqhkiG9w0BCQEWCG1zQHh4LmNu
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKgOibG5uYj2r/X8ufd2ghybJch/wn3f
2xJUzRHK1C543qSLHopGhZWm15Qk6U54Z23gQhlLSpeeSVf+DvverAMCAwEAATAN
BgkqhkiG9w0BAQUFAANBACY5VH25xW+q8WZ0QtXGbnMfOCGgP0zjANwZEon52krI
TxaFrzHYckN2lKmokiNWiz8jtfMMNwHnJg36O7r8oLU=
-----END CERTIFICATE-----
```

按照配置文件将这些证书复制到相应的位置

不要嫌上面的步骤麻烦（我们完全可以直接生成到 demoCA/private/目录下），目的在于说明一个不知道怎么说明的问题（别嫌我绕口，在授课的过程中，我使用直接生成的方法，很多学生不明白，但是使用生成到其他的位置，然后再复制却能明白，我不知道这是为什么）。

```
[root@mail ssl]# pwd
/usr/local/openssl/ssl
[root@mail ssl]# cp private/ca.key  demoCA/private/cakey.pem
[root@mail ssl]# cp private/ca.crt  demoCA/cacert.pem
```

创建证书撤销列表

```
[root@mailssl]#/usr/local/openssl/bin/opensslca-gencrl-outdemoCA/crl/ca.
crl-crldays 7
Using configuration from /usr/local/openssl/ssl/openssl.cnf
unable to load number from ./demoCA/crlnumber
error while loading CRL number
26421:error:0D066091:asn1 encoding routines:a2i_ASN1_INTEGER:odd number of
chars:f int.c:162:
```

这个错误的原因在于 **crlnumber** 文件中没有数值，随便添加一个：

```
[root@mail demoCA]# echo 2000 > crlnumber
```

同样，在下面的 serial 文件也一样，因此需要同样的处理，例如，echo 1000 > serial。

```
[root@mailssl]#/usr/local/openssl/bin/opensslca-gencrl-outdemoCA/crl/ca.
crl-crldays 7
Using configuration from /usr/local/openssl/ssl/openssl.cnf
[root@mail ssl]# cd demoCA/crl
[root@mail crl]# more ca.crl
-----BEGIN X509 CRL-----
MIIBBTCBsAIBATANBgkqhkiG9w0BAQUFADBtMQswCQYDVQQGEwJDTjELMAkGA1UE
CBMCYmoxEDAOBgNVBAcTB2JlaWppbmcxCzAJBgNVBAoTAnBiMQwwCgYDVQQLEwNw
YmExCzAJBgNVBAMTAnltMRcwFQYJKoZIhvcNAQkBFghtc0B4eC5jbhcNMTEwODA5
MDA0NDEzWhcNMTEwODE2MDA0NDEzWqAPMA0wCwYDVR0UBAQCAiAAMA0GCSqGSIb3
```

```
DQEBBQUAA0EAH7Evs9Ch2UfdcgWGFU+x2HG5LQPYjw4MtnoTPHUq+B7QI6SNjC0u
PlgKQ1pgYmApVU8/nHxRXf7snY36zyhWJA==
-----END X509 CRL-----
```

2. 生成服务器端证书

生成服务器私钥

```
[root@mailssl]#/usr/local/openssl/bin/opensslgenrsa-outprivate/server.key
Generating RSA private key, 512 bit long modulus
..............++++++++++++
....++++++++++++
e is 65537 (0x10001)
[root@mail ssl]# more private/server.key
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBAM06k1kAWN5+yvfAsCUQCK1ZjK543IFtOH2Il0Up/cstxkFL81P9
xf3JEqJht3bdDwgTy/n7QHRaPhGfQz0msqkCAwEAAQJABfDBHR1mdSJEa8F3F8uM
BMCH0vEKqPBpZzDR+B2MDBp8wgbOSxM96z+YUr3cY8boCYZD4wSdYzUZxjH56a8v
gQIhAP0BXMG19Dm1tPF4OBXrqafEFmQzvY3FD7QmY0GoL3rZAiEAz6hxyDzNF1lI
abjqL9RwSnaOJrC3XjrmAOef2sMa9FECIQCAckidL7sl8VJDpT0UI+il+69Cf+Ik
L3+hfju2AaTZQQIgRk6kv4P+eHJNNyZZe/o0TrHXLThVzTyp1RKXJ+04+hECIQC+
RYl3SmWsqbG4aBPabgmZPkPsbyi6lL409DZaxCUvjw==
-----END RSA PRIVATE KEY-----
```

生成证书请求

```
[root@mail ssl]# /usr/local/openssl/bin/openssl req -new -key
private/server.key -out private/server.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:bj
Locality Name (eg, city) []:tz
Organization Name (eg, company) [Internet Widgits Pty Ltd]:pb
Organizational Unit Name (eg, section) []:pbz
Common Name (eg, YOUR name) []:dvf
Email Address []:cam@xx.cn
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:dfgdfggx
An optional company name []:vvn
```

```
[root@mail ssl]# more private/server.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIBUjCB/QIBADBqMQswCQYDVQQGEwJDTjELMAkGA1UECBMCYmoxCzAJBgNVBAcT
AnR6MQswCQYDVQQKEwJwYjEMMAoGA1UECxMDcGJ6MQwwCgYDVQQDEwNkdmYxGDAW
BgkqhkiG9w0BCQEWCWNhbUB4eC5jbjBcMA0GCSqGSIb3DQEBAQUAA0sAMEgCQQDN
OpNZAFjefsr3wLAlEAitWYyueNyBbTh9iJdFKf3LLcZBS/NT/cX9yRKiYbd23Q8I
E8v5+0B0Wj4Rn0M9JrKpAgMBAAGgLjAVBgkqhkiG9w0BCQIxCBMGMTIzNDU2MBUG
CSqGSIb3DQEJBzEIEwYxMjM0NTYwDQYJKoZIhvcNAQEFBQADQQA9hAIhKWdsPBrA
Ljt6XiWCyS1TbeRFmh6Qi0BVm6z5roqNrav3nVmKQ+fD+3hbHL9+ixBf2mMASdZx
zqmaP59J
-----END CERTIFICATE REQUEST-----
```

CA 签名

```
[root@mail ssl]# /usr/local/openssl/bin/openssl ca -in private/server.csr
-cert private/ca.crt -keyfile private/ca.key -out demoCA/newcerts/server.crt
Using configuration from /usr/local/openssl/ssl/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 4096 (0x1000)
Validity
Not Before: Aug  9 01:15:32 2011 GMT
Not After : Aug  8 01:15:32 2012 GMT
Subject:
countryName   = CN
stateOrProvinceName   = bj
organizationName  = pb
organizationalUnitName= pbz
commonName= dvf
emailAddress  = cam@xx.cn
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
E6:0B:94:78:30:88:ED:99:87:C2:5B:7F:ED:BF:6E:8C:05:88:94:43
X509v3 Authority Key Identifier:
DirName:/C=CN/ST=bj/L=beijing/O=pb/OU=pba/CN=ym/emailAddress=ms@xx.cn
serial:8E:2F:3D:83:B5:34:87:20

Certificate is to be certified until Aug  8 01:15:32 2012 GMT (365 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

注意最后一行“Data Base Updated”。

我们现在看一下 demoCA 目录：

```
[root@mail demoCA]# tree
.
|-- cacert.pem
|-- certs
|-- crl
|   '-- ca.crl
|-- crlnumber
|-- crlnumber.old
|-- index.txt
|-- index.txt.attr
|-- index.txt.old
|-- newcerts
|   |-- 1000.pem
|   '-- server.crt
|-- private
|   '-- cakey.pem
|-- serial
'-- serial.old

4 directories, 12 files
```

文件名称及文件内容都发生了变化。有关这些内容就不再多叙述了，毕竟我们不是专门讲述证书管理。

3. 生成客户端证书

创建一个存放客户端证书的目录：

```
[root@mail demoCA]# mkdir users
```

生成 user1 的私钥

```
[root@mailssl]#/usr/local/openssl/bin/opensslgenrsa-des3-outdemoCA/users
/user1.key 1024
Generating RSA private key, 1024 bit long modulus
...................++++++
........++++++
e is 65537 (0x10001)
Enter pass phrase for demoCA/users/user1.key:
Verifying - Enter pass phrase for demoCA/users/user1.key:
```

```
[root@mail ssl]# more demoCA/users/user1.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,EED34BA98CDAD995

3ERbJPp5i/qU3oKAnqBrYjT0QTI/7g3yv38WR1AfZFzZj2VbVp+p/JKoiPUojw14
erOY0e/tiaxSwrcdaPNUlzl2KWIwYlkUUft59Yky4KpWgsyhYQV350ICmJRKfJos
NDfT/+rDJtBrDAIxr9SM4mA9XaPf7coQGWRUcNxkRR2tp8A3DnPKfVmTzVr0738q
wN1yqn6H4KY9Po/PpalFUPLMYOQFP09q4tT3TJOOFDNeERZ0a/olKi6v5Un9u8t+
0DQ62iRcoXhk8/vQVAjaIC5hQkuXuGoyzIdnQQSK+x0OUnmxaF/buncMQ/IKlNrb
PIKw7Zlzjh8KPwFSZ1Zqj2efZlPRHHOkNfMdpXAbk1nYFJp1opc0SCD/WjeslEm4
NVvZ+uLxImTnl8NGv0Mj2Qc3nmsu7hm0nvjuUWn4CIbeojamG2eRSGOtLSnO2Vm8
x4sVPB0Yv2po7Z7J2ukvL52jzzIb3VsDKr2n3uDrZp2Kd4wFibuoewkRlnjfldT+
L8WsSUi3190tT4Q0x+JaBYvSQ20POxdSe2xsz+OEp/pG0+Hi0ieJTkcWWDaLqIB8
11zdjY4/qFL6P1nw/U6TD0Wb4cQkkJ93akjrYmQ2VTOgrqlhTjYzUpY5dwnLfHIx
3Qs6idN3JgityrNAO8IxV5t21HGY2o2r6CQl9Lw0x2LJfUdBrDuXYdcFWpRBWmXN
q27IiGI1zEkR4oFKe2WjsLrCqhrPQFhriS9kFMi09GJN3q5zMxUQj+fO00oQNjHf
gihzgaaqy2Ic81pdBthe0GbAXak28zSz5oHbRPOf9H/25DJdXDUJog==
-----END RSA PRIVATE KEY-----
```

生成证书请求

```
[root@mailssl]#/usr/local/openssl/bin/opensslreq-new-keydemoCA/users/use
r1.key -out demoCA/users/user1.csr
Enter pass phrase for demoCA/users/user1.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:bj
Locality Name (eg, city) []:sfj
Organization Name (eg, company) [Internet Widgits Pty Ltd]:pb
Organizational Unit Name (eg, section) []:bpz
Common Name (eg, YOUR name) []:user1
Email Address []:user1@xx.cn

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:hol3hjk
An optional company name []:ff
[root@mail ssl]# more demoCA/users/user1.csr
```

```
-----BEGIN CERTIFICATE REQUEST-----
MIIB2zCCAUQCAQAwbzELMAkGA1UEBhMCQ04xCzAJBgNVBAgTAmJqMQwwCgYDVQQH
EwNzZmoxCzAJBgNVBAoTAnBiMQwwCgYDVQQLEwNicHoxDjAMBgNVBAMTBXVzZXIx
MRowGAYJKoZIhvcNAQkBFgt1c2VyMUB4eC5jbjCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEA0cFlgx3IRX38z+7BT74s08bO0UyqBSx5WaAmLJgxGrNFkhn0CdFz
F563AXZjmDFPoaxk7fig1jSrTm0aothNqRMMUT/PXrPyCOXl5N2mInLRs7o5AK44
bQwI3Y/GFsAmt3MY7W+p3WCC5slIuS1vWHudcXwKhItAMwya8EyKBqsCAwEAAaAs
MBMGCSqGSIb3DQEJAjEGEwQxMjM0MBUGCSqGSIb3DQEJBzEIEwYxMjM0NTYwDQYJ
KoZIhvcNAQEFBQADgYEAS/fwwtZxPZP7Jr3vgrR9bhgHZdcn8WgrwFzOqt+PfjpY
Iz6jTCC/EKLwBilayZN/0022fFkJDGdlfM07y50DDSNLstYmPHeATLJZz7lr9vtv
c/q5pTak9QOIIffCXlymfa++JlGxVrb/8mSnk1O+pJUCHM4UWMHmaRN3iI09Ho4=
-----END CERTIFICATE REQUEST-----
```

CA 签名

```
[root@mailssl]#/usr/local/openssl/bin/opensslca-indemoCA/users/user1.csr
-cert private/ca.crt -keyfile private/ca.key -out demoCA/users/user1.crt
Using configuration from /usr/local/openssl/ssl/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 4097 (0x1001)
Validity
Not Before: Aug  9 01:54:46 2011 GMT
Not After : Aug  8 01:54:46 2012 GMT
Subject:
countryName   = CN
stateOrProvinceName   = bj
organizationName  = pb
organizationalUnitName= bpz
commonName= user1
emailAddress  = user1@xx.cn
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
41:A3:F2:70:B9:51:E2:43:A0:5A:89:2B:3B:DD:0D:99:AB:3F:EE:8B
X509v3 Authority Key Identifier:
DirName:/C=CN/ST=bj/L=beijing/O=pb/OU=pba/CN=ym/emailAddress=ms@xx.cn
serial:8E:2F:3D:83:B5:34:87:20

Certificate is to be certified until Aug  8 01:54:46 2012 GMT (365 days)
Sign the certificate? [y/n]:y
```

```
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

转换证书格式

将证书转换为浏览器能够识别的 PKCS12 格式。这种格式是二进制，不能再使用文本查看器查看。

```
[root@mailssl]#/usr/local/openssl/bin/opensslpkcs12-export-clcerts-indemoCA/users/user1.crt-inkeydemoCA/users/user1.key-outdemoCA/users/user1.p12
Enter pass phrase for demoCA/users/user1.key:
Enter Export Password:
Verifying - Enter Export Password:
```

我们再看一下 users 目录：

```
[root@mail demoCA]# tree  users/
users/
|-- user1.crt
|-- user1.csr
|-- user1.key
'-- user1.p12

0 directories, 4 files
```

将 user1.p12 发送到客户端，这就是客户端需要的证书文件。

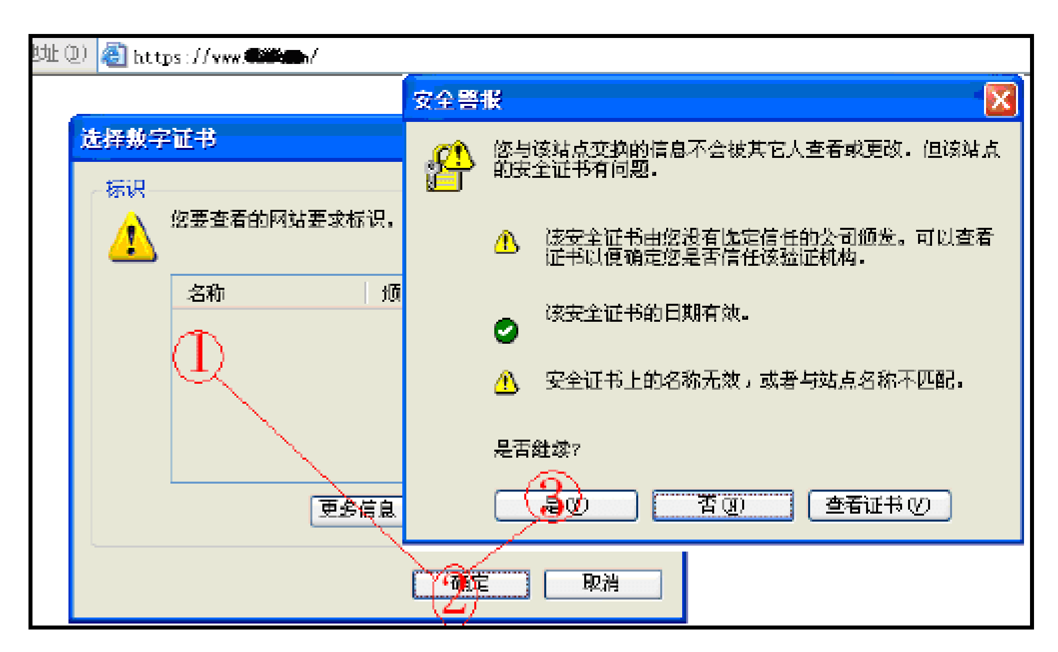
4. 添加服务器配置

```
server {
server name  www.xx.com;
listen 443;
ssl on;
ssl_verify_client on;
ssl_certificate  /usr/local/nginx-1.0.2-ssl/conf/server.crt;
ssl certificate key  /usr/local/nginx-1.0.2-ssl/conf/server.key;
ssl client certificate /usr/local/nginx-1.0.2-ssl/conf/ca.crt;
…
}
```

复制证书

将前面生成的证书，根据 Nginx 配置文件中添加的配置情况将其复制到 Nginx 的 conf 目录下，然后重新启动 Nginx 服务器。

访问测试

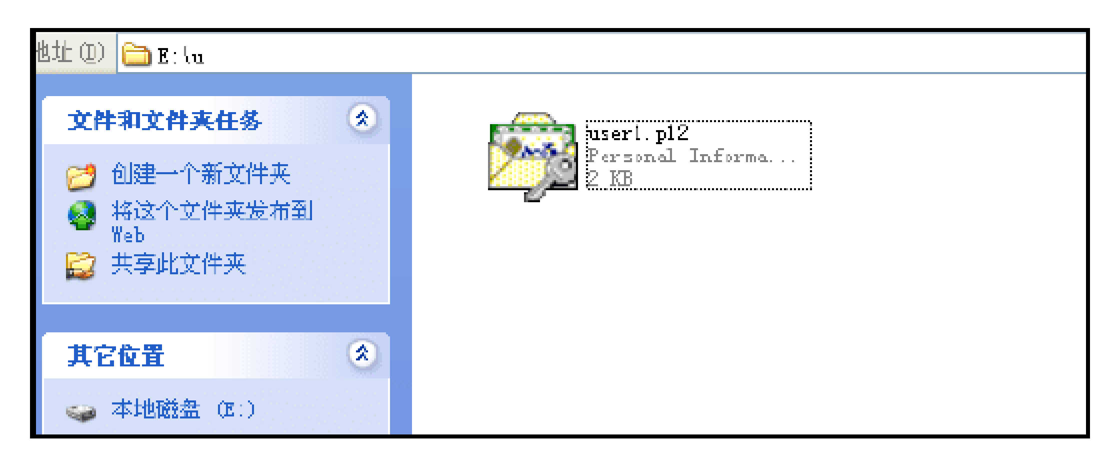


我们看一下在标注“1”的地方没有证书，然后单击 OK 按钮，访问结果如下：



毫无疑问，由于证书没有发送，所以访问失败。

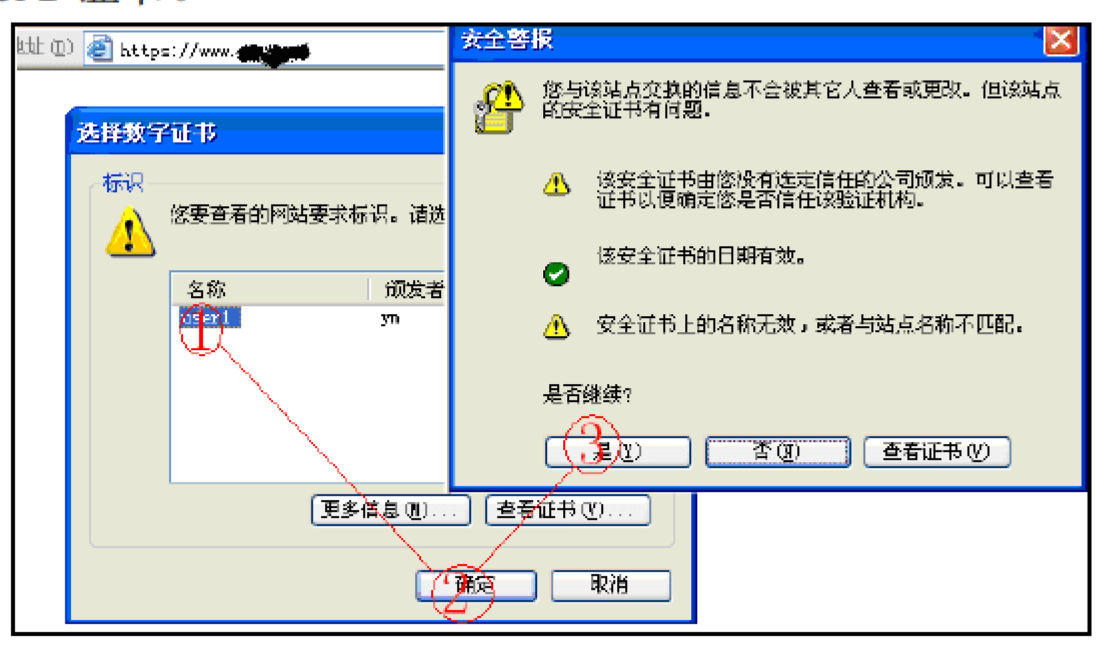
导入客户端证书



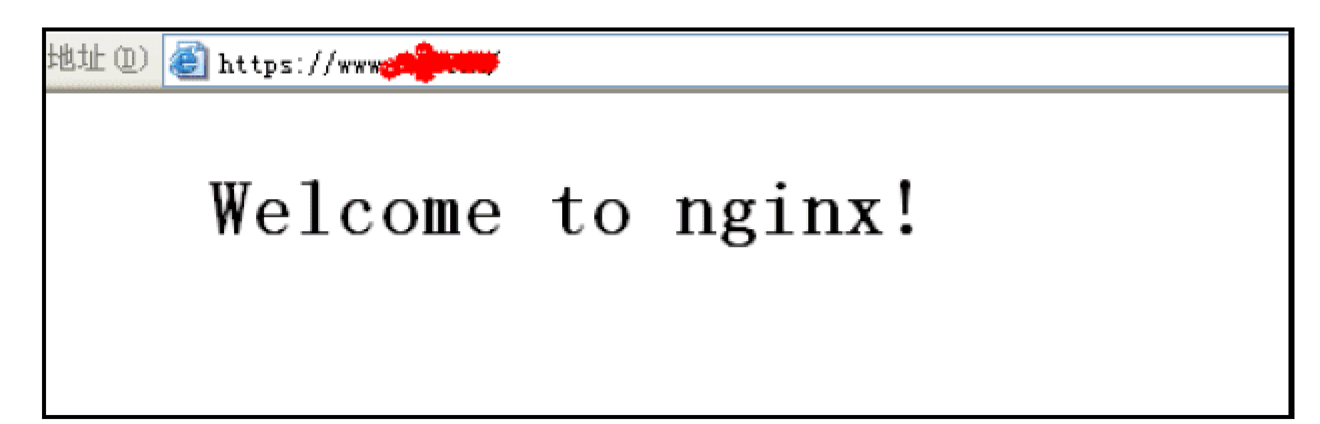
双击该图标就可自动导入。在导入的过程中需要输入密码。

再次访问测试

这次我们选择 user1 证书。



访问结果如下：



很明确，双向验证成功，访问成功。

32.7 HTTPS 服务器优化

SSL 操作会消耗更多的 CPU 资源，在一个多 CPU 系统上，我们应该运行多个 worker 进程：不要超过可用的 CPU 内核数。占用 CPU 最多的操作是 SSL 握手。有两种方法可以将每一个客户端的这种操作降低到最少，一是增加会话缓存，二是延长缓存存放时间。

所有与 worker 的会话都被存储在一个 SSL 会话缓存中，该缓存由指令 ssl_session_cache 指定，1MB 缓存可以存储 4000 个会话。默认的缓存超时为 5 分钟，根据实际情况可以增加该值，通过指令 ssl_session_timeout 来实现。下面是一个简单的、尽可能优化的配置文件，一个四核系统，配置了 10MB 共享会话的缓存，另外也将 ssl_session_timeout 的值设置为 10 分钟：

```
worker_processes  4;

http {
ssl_session_cacheshared:SSL:10m;
ssl_session_timeout  10m;

server {
listen   443;
server name  www.nginx.com;
keepalive_timeout70;

ssl  on;
ssl_certificate  www.nginx.com.crt;
ssl_certificate_key  www.nginx.com.key;
ssl protocolsSSLv3 TLSv1;
ssl ciphers  HIGH:!ADH:!MD5;
…
```

某些浏览器可能会抱怨一些由众所周知（就是由权威认证机构颁发）的证书认证授权机构签发的证书，而其他的浏览器可以完全没问题地接受证书。

第 33 章　监控 Nginx 的工作状态

该模块提供了获取 Nginx 工作状态的功能，默认情况下该模块是不会被编译包含在内，如果需要该模块，那么需要指定--with-http_stub_status_module，例如:

```
[root@mailconf]#./configure--prefix=/usr/local/nginx-0.8.54--with-http s
tubstatus_module
```

1. 配置示例

```
location /nginx status {
  # copied from http://blog.kovyrin.net/2006/04/29/monitoring-nginx-with
-rrdtool/
  stub status on;
  access_log   off;
  allow SOME.IP.ADD.RESS;
  deny all;
}
```

2. 指令

该模块只提供了一个指令，那就是 stub_status。

指令名称：stub_status

语法：stub_status on

默认值：None

使用环境：location

功能：对该 location 启用状态监控。stub status 指令提供的状态内容类似于 mathopd 的状态页，纯文本信息类似于：

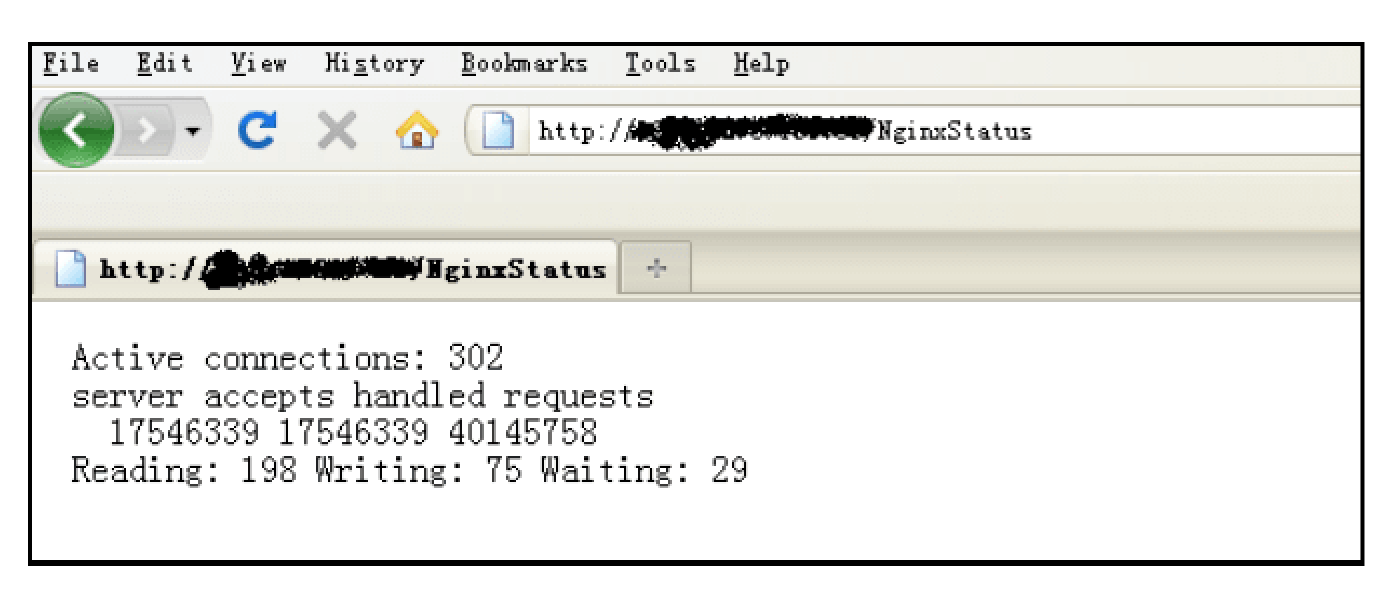
```
 Active connections: 291
server accepts handled requests
 16630948 16630948 31070465
Reading: 6 Writing: 179 Waiting: 106
```

3. 使用实例

如果在安装 Nginx 时选择了--with-http_stub_status_module 模块，那么在配置文件中会有以下配置：

```
location /NginxStatus {
  stub status on;
}
```

当然，根据需要这个 URI 还可以修改。下面是访问：



正如前面所说，它会报告一些当前的工作状态。下面我们看一下这些指标：

- Active connections：活动连接数，包括后台的连接数；
- server accepts handled requests：服务器接收处理的请求数，接收了 16630948 个连接，成功处理了 16630948 个连接（没有一个连接关闭，都在接受请求），成功处理了 31070465 个请求，平均每个请求处理 1.8 个请求，这个数值是由 31070465 除以 16630948 得出的。
- Reading：Nginx 读取请求头信息
- Writing：Nginx 读取请求体，处理请求或者是向客户端写响应信息。
- Waiting：keep-alive 连接，通常是活动的连接，即 Reading 和 Writing。

第 34 章　使用 empty_gif

如果我们想在站点出错的时候，留给用户的不是一个专业术语，而又可能对用户来说无意义的一个错误代码，那么你可以使用 empty_gif 来敷衍一下；如果想检查服务器的存活情况，也可以使用 empty_gif，以及提供其他一些应答。

empty_gif 模块在内存中保存一个可以快速传递的 1×1 透明 GIF，而不用再去读取磁盘。

1. 配置示例

```
location = /_.gif {
  empty_gif;
}
```

2. 指令

empty_gif 模块只有一个指令那就是 empty_gif。

指令名称：empty_gif

语法：empty_gif

默认值：n/a

使用字段：location

功能：从内存提供一个 1×1 透明 GIF 文件。

3. 使用实例

类似下面的这种情况我们见过：



> 如果是因为系统频繁地打开文件所造成，那么解决方法在相关的章节中有说明，但是如果仍然超出限制呢？将系统的设置调到最大限制当然好，可服务器不一定承受得了，弄不好还有可能崩溃，但留给客户端的访问响应为 500 固然也不好，碰上不懂行的（其实大多数访问者都不会懂这种代码）就不再访问了，其实可能刷新或者稍微等会就又可以访问了。

添加配置

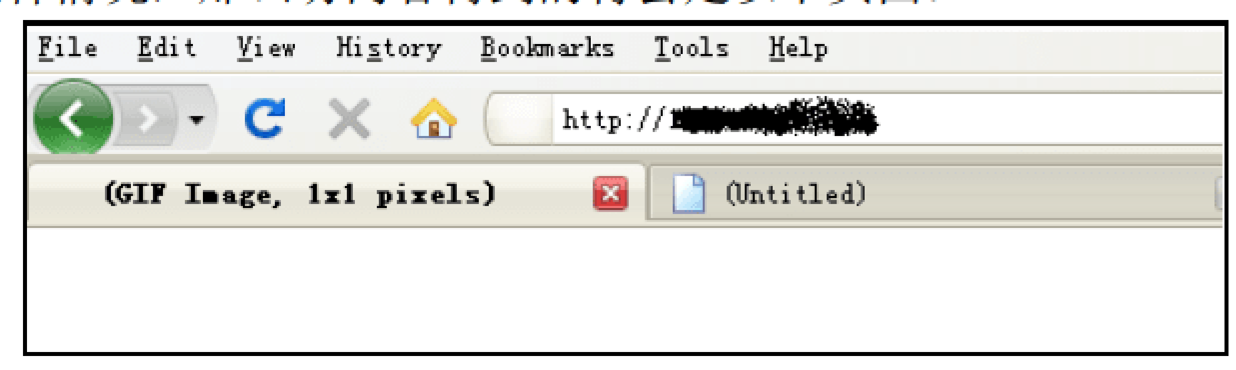
上面的问题我们可以通过以下配置解决。在 Nginx 的配置文件中添加以下代码：

```
error_page  500 /1.gif;
location = /1.gif {
```

```
    empty_gif;
  }
```

访问测试

如果再出现这种情况，那么访问者得到的将会是以下页面：



空白页没有任何内容，同时它的源码也不能看。同理，403、404 等，也可以这么做。

其他的例子就不再列举了，可能还有其他的功能吧。

第 35 章　Nginx 实现对响应体内容的替换

该模块能够搜索和替换 Nginx 响应体中的文本内容。这个模块在默认安装 Nginx 时是不会安装的，因此，要想使用该模块，那么需要在./configure 时添加--with-http_sub_module option 选项。

1. 配置示例

```
location / {
  sub_filter  </head>
  '</head><script language="javascript" src="$script"></script>';
  sub_filter_once on;
}
```

2. 指令

该模块提供了 3 条指令。

指令名称：sub_filter

语法：sub_filter text substitution

默认值：none

使用环境：http，server，location

功能：该指令用于在 Nginx 的响应中替代一些文本，即将原有的“text”替换为现有的“substitution”，而不依赖于源数据。内容匹配对大小写不敏感。替代文本可以包含变量，每一个 location 中只能使用一种替换规则。

指令名称：sub_filter_once

语法：sub_filter_once on|off

默认值：sub_filter_once on

使用环境：http，server，location

功能：如果将该指令设置为 off，那么将会允许搜索和替换所有匹配的行。默认情况下仅替换第一个被匹配的行。

指令名称：sub_filter_types

语法：sub_filter_types mime-type [mime-type ...]

默认值：sub_filter_types text/html

使用环境：http，server，location

功能：该指令用于指定 sub_filter 指令应该检测的内容类型。默认只有 text/html。

3. 使用实例

在 Nginx 的配置文件中添加以下配置内容：

```
  http {
    include   mime.types;
    default_type  application/octet-stream;

    sendfileon;

    keepalive timeout  65;

sub_filter  '</head>'  '<style type="text/css">html {filter:progid:DXImage
Transform.Microsoft.BasicImage (grayscale=1) ; }</style></head>';
sub_filter_once on;

  server {
    listen   80;
    server name  localhost;

location / {
  root   html;
  index  index.html index.htm;
    }
  }
  …
  }
```

这是某年用得最多的一个例子，它将所有的页面在 IE 浏览器下访问都变为灰色。

第 36 章　Nginx 的 WebDAV

在 Linux 系统中，我们可以使用 Nautilus 内置的 WebDAV 来进行实际的工作。在这里我们通过 telnet 命令来完成，以便了解工作过程。而实际中则使用某一种语言的函数来完成，这就是程序的事情了，我们的主要目的还是为别人做嫁衣，测试通过就 OK。

> 另外，我们也要多掌握一些相关的知识，比如各种响应代码，以便解决问题（避免不必要的扯皮！你懂的）。

该模块用于添加 HTTP 和 WebDAV 方法 PUT、DELETE、MKCOL、COPY 和 MOVE，由于该模块在默认安装时没有安装，因此，如果需要使用它，那么在./configure 时要添加--with-http_dav_module 选项。

这是从 Nginx（1.0.2 版本）源码中节选 ngx_http_dav_module.c 的部分内容，说明了 DAV 当前支持的方法：

```
static ngx_conf_bitmask_t  ngx_http_dav_methods_mask[] = {
{ ngx_string("off") , NGX_HTTP_DAV_OFF },
{ ngx_string("put") , NGX_HTTP_PUT },
{ ngx_string("delete") , NGX_HTTP_DELETE },
{ ngx_string("mkcol") , NGX_HTTP_MKCOL },
{ ngx_string("copy") , NGX_HTTP_COPY },
{ ngx_string("move") , NGX_HTTP_MOVE },
{ ngx_null_string, 0 }
};
```

1. 配置示例

```
location / {
  root /data/www;
  client_body_temp_path  /data/client_temp;

  dav methods  PUT DELETE MKCOL COPY MOVE;

  create full put path   on;
  dav access group:rw  all:r;

  limit_except  GET {
allow  192.168.1.0/32;
deny   all;
  }
}
```

2. 指令

该模块提供了以下 3 条指令。

指令名称：dav_access

语法：dav_access user:permissions [users:permissions] ...

默认值：dav_access user:rw

使用环境：http，server，location

功能：该指令用于为文件和目录设定访问权限，例如：

```
dav_access user:rw group:rw all:r;
```

如果为 groups 或 all 设定了任何权限，那么就没有必要为 user 指定权限了，例如：

```
dav_access group:rw all:r;
```

指令名称：dav_methods

语法：dav_methods [off|put|delete|mkcol|copy|move] ...

默认值：dav_methods off

使用环境：http，server，location

功能：该指令用于指定 HTTP 和 WebDAV 方法，如果将该指令的值设置为 off，那么将会禁用所有的方法，并且忽略剩余的参数。

对于 PUT 方法，目标文件必须与作为临时存储文件存储的目录位于同一个分区上（由指令 client_body_temp_path 在 location 部分设定）。

当一个文件被使用 PUT 方法创建的时候，会通过 Date 头来修改日期。

指令名称：create_full_put_path

语法：create_full_put_path on|off

默认值：create_full_put_path off

使用环境：http，server，location

功能：在默认情况下，PUT 方法只能够在存在的目录中创建文件，如果将该指令的值设置为 on，那么 PUT 指令将会创建所有的中间目录。

3. 使用实例

在 Nginx 的配置文件中添加以下内容：

```
location / {
  root /web root;
  client body temp path  /web root/client temp;

  dav_methods  PUT DELETE MKCOL COPY MOVE;

  create_full_put_path   off;
  dav access group:rw  all:r;

  limit except  GET {
allow  192.168.3.0/24;
```

```
deny   all;
  }
}
```

目录结构：

```
[root@mail web root]# tree
.
|-- a
|-- client_temp
'-- index.html

2 directories,1 files
```

执行 COPY 命令：

```
[root@jh-share bar]# telnet 192.168.3.139 80
Trying 192.168.3.139...
Connected to www.xx.com (192.168.3.139).
Escape character is '^]'.
COPY /index.html HTTP/1.1
Host: 192.168.3.139
Destination: http://192.168.3.139/a/index.html

HTTP/1.1 204 No Content
Server: nginx/1.0.2
Date: Thu, 01 Sep 2011 00:01:55 GMT
Connection: keep-alive

Connection closed by foreign host.
```

查看 Nginx 的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.140 - - [01/Sep/2011:08:01:55 +0800] "COPY /index.html HTTP/1.1"
204 0 "-" "-"
```

状态为 204。

查看现在的目录结构：

```
[root@mail web_root]# tree
.
|-- a
|   '-- index.html
|-- client temp
'-- index.html

2 directories, 2 files
```

文件 COPY 成功。

在 COPY 操作中可能会出现的状态代码：

状态代码	含　义
201	创建完成（Created）。复制的资源被成功操作完成
204	操作成功（No Content）。源资源被成功地复制到一个已经存在的目标资源中（即目标文件存在情况下的复制操作完成）
207	多个响应状态（Multi-Status）。在 XML 体内发现多个响应代码
403	禁止操作（Forbidden）。源资源的 URI 和目标 URI 相同
409	出现争执情况（Conflict）。资源不能在指定的 URI 下创建，直到一个或者多个中间目录被创建
412	先决条件失败（Precondition Failed）。可能是 Overwrite 头是“F”，并且目标资源状态不为空（null），也可能是该方法（就是 MOVE）使用 Depth: 0 处理
423	锁定（Locked）。目标资源被锁定
502	网关错误（Bad Gateway）。复制操作中，目标文件位于不同的服务器上，而该服务器拒绝接受该资源
507	空间不足（Insufficient Storage）。目标资源没有足够的存储空间

执行 DELETE 命令：

```
[root@jh-share bar]# telnet 192.168.3.139 80
Trying 192.168.3.139...
Connected to www.xx.com (192.168.3.139).
Escape character is '^]'.
DELETE /a/index.html HTTP/1.1
Host: 192.168.3.139

HTTP/1.1 204 No Content
Server: nginx/1.0.2
Date: Thu, 01 Sep 2011 00:22:59 GMT
Connection: keep-alive


Connection closed by foreign host.
```

查看 Nginx 的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.140 - - [01/Sep/2011:08:22:59 +0800] "DELETE /a/index.html HTTP/1.1" 204 0 "-" "-"
```

查看现在的目录结构：

```
[root@mail web_root]# tree
.
|-- a
|-- client temp
'-- index.html

2 directories,1 files
```

在 DELETE 操作中可能会出现的状态代码：

状态代码	含 义
204	已无内容（No Content）。成功执行响应代码
423	锁定（Locked）。目标资源被锁定

执行 PUT 命令：

```
[root@jh-share bar]# telnet 192.168.3.139 80
Trying 192.168.3.139...
Connected to www.xx.com (192.168.3.139).
Escape character is '^]'.
PUT /a/xx.html HTTP/1.1
Host: 192.168.3.139
Content-Length: 9

1234567
HTTP/1.1 201 Created
Server: nginx/1.0.2
Date: Thu, 01 Sep 2011 01:10:53 GMT
Content-Length: 0
Location: http://192.168.3.139/a/xx.html
Connection: keep-alive

Connection closed by foreign host.
```

查看现在的目录结构：

```
[root@mail web_root]# tree
.
|-- a
|   '-- xx.html
|-- client_temp
'-- index.html

2 directories, 2 files
```

在 PUT 操作中可能会出现的状态代码：

状态代码	含 义
201	创建完成（Created）。新的文件被成功创建
204	已无内容（No Content）。成功修改了已经存在的资源
501	命令没有生效（Not Implemented）。不能按照 URI 中指定的去创建或者修改资源

执行 MOVE 命令：

```
[root@jh-share bar]# telnet 192.168.3.139 80
Trying 192.168.3.139...
Connected to www.xx.com (192.168.3.139).
Escape character is '^]'.
MOVE /a/xx.html HTTP/1.1
```

```
Host:192.168.3.139
Destination: http://192.168.3.139/a/index.html

HTTP/1.1 204 No Content
Server: nginx/1.0.2
Date: Thu, 01 Sep 2011 01:35:58 GMT
Connection: keep-alive

Connection closed by foreign host
```

查看 Nginx 的访问日志：

```
[root@mail logs]# tail -f access.log
192.168.3.140 - - [01/Sep/2011:09:35:58 +0800] "MOVE /a/xx.html HTTP/1.1"
204 0 "-" "-"
```

查看现在的目录结构：

```
[root@mail web_root]# tree
.
|-- a
|   '-- index.html
|-- client temp
'-- index.html

2 directories, 2 files
```

在 MOVE 操作中可能会出现的状态代码：

状态代码	含　义
201	创建完成（Created）。资源移动成功，并且新的资源在指定的 URI 中创建
204	已无内容（No Content）。资源被成功移动到一个已经存在的目标 URI
403	禁止操作（Forbidden）。源资的 URI 和目标 URI 相同
409	出现争执情况（Conflict）。资源不能够在指定的目标建立，直到一个或者多个中间路径被创建
412	先决条件失败（Precondition Failed）。可能是 Overwrite 头是“F”，并且目标资源状态不为空（null），也可能是该方法（就是 MOVE）使用 Depth: 0 处理
423	锁定（Locked）。目标资源被锁定
502	网关错误（Bad Gateway）。在移动操作中，目标文件位于不同的服务器上，而该服务器拒绝接受该资源

执行 MKCOL 命令：

```
[root@jh-share bar]# telnet 192.168.3.139 80
Trying 192.168.3.139…
Connected to www.xx.com (192.168.3.139).
Escape character is '^]'.
MKCOL /a/b/ HTTP/1.1
Host: 192.168.3.139
Content-Type: text/html
```

```
Content-Length: 2

HTTP/1.1 415 Unsupported Media Type
Server: nginx/1.0.2
Date: Thu, 01 Sep 2011 02:36:54 GMT
Content-Type: text/html
Content-Length: 194
Connection: keep-alive

<html>
<head><title>415 Unsupported Media Type</title></head>
<body bgcolor="white">
<center><h1>415 Unsupported Media Type</h1></center>
<hr><center>nginx/1.0.2</center>
</body>
</html>
Connection closed by foreign host
```

在 MKCOL 操作中可能会出现的状态代码：

状态代码	含　义
201	创建完成（Created）。目录创建操作成功完成
401	访问被拒绝（Access Denied）。资源请求需要认证或是认证失败
403	禁止访问（Forbidden）。服务器不允许在指定的位置创建目录，或者是指定请求的 URI 父目录存在，但是它不接受成员
405	方法不允许（Method Not Allowed）
409	出现争执情况（Conflict）。资源不能够在指定的目标建立，直到一个或者多个中间路径被创建
415	不支持的媒体类型（Unsupported Media Type）。请求体的类型不被服务器支持
507	空间不足（Insufficient Storage）。目标资源没有足够的存储空间

第 37 章　Nginx 的 Xslt 模块

Xslt 模块是一个过滤器，可以借助一个或者多个 Xslt 模块转化 XML 响应。该模块从 0.7.8 版本开始的 Nginx 服务器提供，但是在默认安装中并没有选择这个模块，因此，如果想使用该模块，那么在编译安装 Nginx 时需要指定--with-http_xslt_module 选项。

1. 配置示例

```
location / {
  xml_entities   /site/dtd/entities.dtd;
  xslt_stylesheet/site/xslt/one.xslt   param=value;
  xslt_stylesheet/site/xslt/two.xslt;
}
```

2. 指令

Xslt 模块提供了以下 3 条指令。

指令名称：xml_entities

语法：xml_entities <path>

默认值：no

使用环境：http，server，location

功能：指定 DTD 文件，描述符号元素（xml 条目）。该文件在编译配置文件阶段完成。由于技术原因，可能在 xml 中指定的条目不会被处理，因此它们会被忽略，但是这个被指定的 DTD 文件将会被替代使用，在该文件中没有必要描述 XML 的结构，而只需要声明基本的 XML 文档标记即可。例如：

```
<! ENTITY of nbsp "  ">
```

指令名称：xslt_stylesheet

语法：xslt_stylesheet template [parameter[[parameter...]] default: no

使用环境：http，server，location

功能：通过 parameter 参数指定 Xslt 模板。模板在编译配置文件阶段完成。参数传递值的格式为：param=value

可以在一行指定一个参数，也可以在一行指定多个参数，但是要使用冒号（:）隔开。如果参数本身包含冒号（:）字符，那么要使用%3A 进行转义。此外，如果 value 中包含非字母数字字符，那么 libxslt 要求对这些字符串使用单引号或者是双引号，例如：

```
param1='http%3A//www.example.com': param2=value2
```

另外，还可以使用变量作为参数，例如：

```
location / {
xslt stylesheet /site/xslt/one.xslt
$arg xslt params
param1='$value1': param2=value2
```

```
param3=value3;
  }
```

可以指定多个模块，在这种情况下，它们将会按照在配置文件中的顺序被连接在一起。

指令名称：xslt_types

语法：xslt_types mime-type [mime-type...]

默认值：xslt_types text/xml

使用环境：http，server，location

功能：该指令用于设定除了处理 text/xml 之外的 MIME 类型。如果 Xslt 输出模式是 HTML，那么响应的 MIME 类型将会改变为 text/HTML。

3. 使用实例

我没有使用过该模块，据说有人使用它来做搜索引擎优化。

第 38 章　Nginx 的基本认证方式

如同 Apache 服务器一样，Nginx 也提供了基本身份认证。Nginx 的基本认证由 HttpAuthBasicModule 模块来完成，我们通过使用该模块来保护站点或者站点的某一部分，以便通过用户名和密码来访问保护的资源。下面我们来看一下这个模块。

1. 配置示例

```
location / {
  auth_basic"Restricted";
  auth_basic_user_file conf/htpasswd;
}
```

2. 指令

Apache 模块提供了两个指令。下面我们来看一下这两个指令的功能。

指令名称：auth_basic

语法：auth_basic realm | off

默认值：auth_basic off

使用环境：http，server，location，limit_except

功能：该指令用于在本级别（或者说从本位置开始）启用 Nginx 的基本身份认证功能。如果将它的值设置为 off，那么将不会继承上一级的设置（即用户名和密码），即它会覆盖掉上一级的设置；可以设置的另一种可能就是设置一个字符串，这个字符串可以是有意义的，也可以是无意义的，但是它最终会在登录框中出现，它代表的是一个执行权限的权限域。

指令名称：auth_basic_user_file

语法：auth_basic_user_file file

默认值： none

使用环境：http，server，location，limit_except

功能：该指令用于为认证域（就是前面说的从某一个位置开始）设置密码文件。从 Nginx 的 0.6.7 版本开始，该文件的位置不再依赖于安装 Nginx 指定的 prefix 目录了，而是依赖于 Nginx 的配置文件 nginx.conf。

3. 密码文件的格式

同 Apache 的密码文件格式一样，格式为：

```
用户:密码:注解
```

例如：

```
user:pass
user2:pass2:comment
user3:pass3
```

38.1 生成密码

密码文件中的密码部分必须是通过 crypt（3）函数编码加密的。如果安装了 Apache，那么可以使用 Apache 的 htpasswd 命令来生成，但是要注意该命令默认使用的是 MD5 算法，而非 CRYPT，因此要使用-d 选项，以便强制使用 CRYPT 加密方式。

1. 使用 Apache 的 htpasswd 命令

例如，我们要使用以下用户：

```
[root@flv conf]# vi passwd1.txt
student1: student1: wanghui
student2: student2: liuran
student3: student3: liuling
```

```
[root@flv conf]# vi passwd2.txt
student4: student1: shixi
```

添加这些用户：

```
[root@flv conf]# /usr/bin/htpasswd  -dbc  passwd1  student1 student1
Adding password for user student1
[root@flv conf]# /usr/bin/htpasswd  -db  passwd1  student2 student2
Adding password for user student2
[root@flv conf]# /usr/bin/htpasswd  -db  passwd1  student3 student3
Adding password for user student3
```

```
[root@flv conf]# /usr/bin/htpasswd  -dbc  passwd2  student4 student4
Adding password for user student4
```

> 注意，在这里我使用了-b 参数，在实际的使用中考虑到安全的问题，不要这么做。

看一下加密后的密码文件：

```
[root@flv conf]# cat passwd1
student1:JQmo1uo7BqLLs
student2:aWzVVvbYeA3xQ
student3:N7MIoSzN3xbaU
```

我们可以将注解添加在文件中：

```
[root@flv conf]# cat passwd1
student1:JQmo1uo7BqLLs:wanghui
student2:aWzVVvbYeA3xQ:liuran
student3:N7MIoSzN3xbaU:liuling
```

2. 使用 Python 的 httppasswd 命令

这个文件在很久以前下载使用过，忘记它的下载地址了，因此就在这里将它全部贴出来了。注意加粗字体部分。另外根据需要还可以对该文件进行修改使用：

```
[root@flv ~]# more httppasswd
#!/usr/bin/python
"""Replacement for htpasswd"""
# Original author: Eli Carter

import os
import sys
import random
from optparse import OptionParser

# We need a crypt module, but Windows doesn't have one by default.  Try to
find
# one, and tell the user if we can't.
try:
import crypt
except ImportError:
try:
import fcrypt as crypt
except ImportError:
sys.stderr.write("Cannot find a crypt module.  "
 "Possibly http://carey.geek.nz/code/python-fcrypt/\n")
sys.exit(1)


def salt():
"""Returns a string of 2 randome letters"""
letters = 'abcdefghijklmnopqrstuvwxyz' \
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ' \
  '0123456789/.'
return random.choice(letters) + random.choice(letters)


class HtpasswdFile:
"""A class for manipulating htpasswd files."""

def __init__(self, filename, create=False):
self.entries = []
self.filename = filename
if not create:
if os.path.exists(self.filename):
```

```
self.load ()
else:
raise Exception ("%s does not exist" % self.filename)

def load (self) :
"""Read the htpasswd file into memory."""
lines = open (self.filename, 'r') .readlines ()
self.entries = []
for line in lines:
username, pwhash = line.split (':')
entry = [username, pwhash.rstrip () ]
self.entries.append (entry)

def save (self) :
"""Write the htpasswd file to disk"""
open (self.filename, 'w') .writelines (["%s:%s\n" % (entry[0], entry[1])
 for entry in self.entries])

def update (self, username, password) :
"""Replace the entry for the given user, or add it if new."""
pwhash = crypt.crypt (password, salt () )
matching entries = [entry for entry in self.entries
if entry[0] == username]
if matching_entries:
matching_entries[0][1] = pwhash
else:
self.entries.append ([username, pwhash])

def delete (self, username) :
"""Remove the entry for the given user."""
self.entries = [entry for entry in self.entries
if entry[0] != username]


def main () :
"""%prog [-c] -b filename username password
Create or update an htpasswd file"""
# For now, we only care about the use cases that affect tests/functional.py
parser = OptionParser (usage=main.__doc__)
parser.add_option('-b', action='store_true', dest='batch', default=False,
help='Batch mode; password is passed on the command line IN THE CLEAR.'
)
parser.add_option    (    '-c',     action='store_true',     dest='create',
```

```
default=False,
    help='Create a new htpasswd file, overwriting any existing file.')
    parser.add_option ('-D', action='store_true', dest='delete_user',
    default=False, help='Remove the given user from the password file.')

    options, args = parser.parse_args ()

    def syntax_error (msg) :
    """Utility function for displaying fatal error messages with usage
    help.
    """
    sys.stderr.write ("Syntax error: " + msg)
    sys.stderr.write (parser.get_usage () )
    sys.exit (1)

    if not options.batch:
    syntax_error ("Only batch mode is supported\n")

    # Non-option arguments
    if len (args)  < 2:
    syntax_error ("Insufficient number of arguments.\n")
    filename, username = args[:2]
    if options.delete_user:
    if len (args)  != 2:
    syntax_error ("Incorrect number of arguments.\n")
    password = None
    else:
    if len (args)  != 3:
    syntax_error ("Incorrect number of arguments.\n")
    password = args[2]

    passwdfile = HtpasswdFile (filename, create=options.create)

    if options.delete user:
    passwdfile.delete (username)
    else:
    passwdfile.update (username, password)

    passwdfile.save ()


    if   name    == '  main   ':
    main ()
```

看一下 httppasswd 的用法：

```
[root@flv ~]# python httppasswd --help
Usage: httppasswd [-c] -b filename username password
Create or update an htpasswd file

Options:
  -h, --help  show this help message and exit
  -b  Batch mode; password is passed on the command line IN THE CLEAR.
  -c  Create a new htpasswd file, overwriting any existing file.
  -D  Remove the given user from the password file.
```

可以看得出，httppasswd 的使用方法与 Apache 的 htpasswd 命令相似，而且它支持的就是 CRYPT 加密方式，因此就不用再指定加密方式了（其实也没得指定！）。

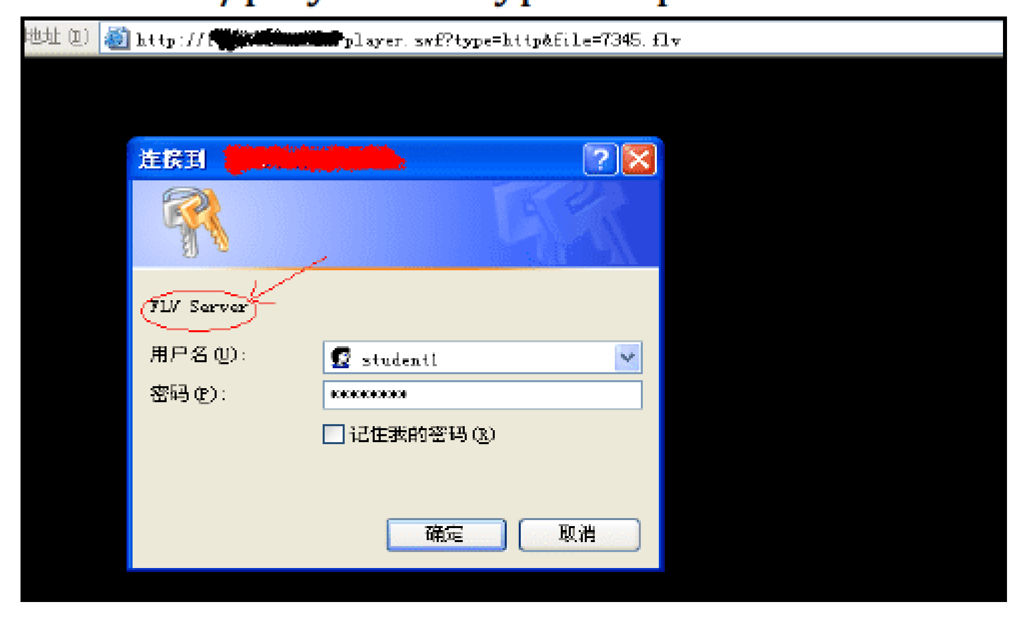
38.2 添加配置

为了测试密码或者是权限域，我们在这里分别对两个 location 设置了使用不同的用户权限：

```
location ~ \.flv$ {
auth basic "FLV Server";
auth basic user file  /usr/local/nginx-1.0.2-mp4-flv/conf/passwd1;
root /var/www/flv;
flv;
}
location ~ \.mp4$ {
auth_basic "MP4 Server";
auth basic user file  /usr/local/nginx-1.0.2-mp4-flv/conf/passwd2;
root /var/www/mp4;
mp4;
  }
```

38.3 访问测试

下面访问 http://www.xx.com/player.swf?type=http&file=7345.flv



可见我们需要输入用户名和密码，输入相应的（就是 passwd1 中）用户名和密码，就可成

功访问。我们看一下 Nginx 的访问日志：

```
    [root@mail log]# tail -f access.log
    192.168.3.248    -    student1    [01/Sep/2011:20:12:27    +0800]    "GET
/7345.flv?start=0 HTTP/1.1" 200 17616730 "-" "Mozilla/5.0 (Windows; U; Windows
NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

在成功访问该网址后，不要关闭这个浏览器，我们在地址栏中输入以下地址 http://www.xx.com/player.swf?type=http&file=4315.mp4：



可以看到，以前的验证不再有效，需要输入新的用户名和密码。同样我们看一下 Nginx 的访问日志：

```
    [root@mail log]# tail -f access.log
    192.168.3.248    -    student4    [01/Sep/2011:20:16:43    +0800]    "GET
/4315.mp4?start=0 HTTP/1.1" 200 7341780 "-" "Mozilla/5.0 (Windows; U; Windows
NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

第 39 章　Nginx 的 cookie

在 Nginx 中提供了一个 ngx_http_userid_module 模块，它的功能就是颁发 cookie，在默认安装中就被选择安装。

该模块用于颁发 cookie，以便于在子请求上标识客户端。为了在日志中记录 cookie 信息，ngx_http_userid_module 还提供了两个变量：$uid_got 和 $uid_set。

> 注意，变量$uid_got 和 $uid_set 在 SSI 中不容易取到，因为 SSI 过滤模块在整个 Nginx 处理链中要早于 userid 模块。

该模块的功能与 Apache 的 mod_uid 模块功能一致。

1. 配置示例

```
userid  on;
userid name uid;
userid_domain   example.com;
userid_path /;
userid_expires  365d;
userid_p3p'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';
```

2. 指令

该模块提供了以下 7 条命令。

指令名称：userid

语法：userid [on|v1|log||off]

默认值：userid off

使用环境：http，server，location

功能：启用或者禁用颁发 cookie 和记录被请求 cookie。可选项如下。

- on：启用版本 2 的 cookies 并且记录它们。
- v1：启用版本 1 的 cookies 并且记录它们。
- log：不发送 cookies，但是记录进入的 cookies。
- off ：不发送 cookies，也不记录到日志。

指令名称：userid_domain

语法：userid_domain [name | none]

默认值：userid_domain none

使用环境：http，server，location

功能：为指定的域签发 cookie，如果将该指令的参数设置为 none，那么将不对任何域名发出 cookie。

指令名称：userid_expires

语法：userid_expires [time | max]

默认值：none

使用环境：http，server，location

功能：为 cookie 设置生存期。该指令是用于为浏览器设置和发送 cookie 生存期，如果设置为 max，那么将会为浏览器设置的生存期是到 31 December，2037，23:55:55 gmt。这个 max 对于一些老的浏览器可能不会被认识。

指令名称：userid_name

语法：userid_name name

默认值：userid_name uid

使用环境：http，server，location

功能：设定 cookie 的名字。

指令名称：userid_p3p

语法：userid_p3p line

默认值：none

使用环境：http，server，location

功能：该指令用于指定 P3P 头的值，它将会与 cookie 一同发送到客户端。

指令名称：userid_path

语法：userid_path path

默认值：userid_path /

使用环境：http，server，location

功能：设置 cookie 的路径。

指令名称：userid_service

语法：userid_service number

默认值：userid_service address

使用环境：http，server，location

功能：设置发布 cookie 的服务器地址。如果没有设置服务器地址，若是版本 1，那么会将 cookies 设置为 0，若是版本 2，那么 cookies 将会设置为服务器的 IP 地址。

3. 使用实例

在 Nginx 服务器中添加以下配置内容：

```
 http {
include   mime.types;
default_type  application/octet-stream;

sendfileon;

keepalive_timeout  65;
```

```
 log_format custom $time_local | $server_name | $request_length | $bytes_sent
| $uid_got | $uid_set;

  server {
 listen   80;
 server name  www.xx.com;

 access_log logs/custom.log custom;

 userid_name uid;
 userid domain   xx.com;
 userid path /;
 userid expires  365d;
 userid p3p   'policyref="/w3c/p3p.xml", CP="CUR ADM OUR NOR STA NID"';

 location / {
   root   html;
   index  index.html index.htm;
 }

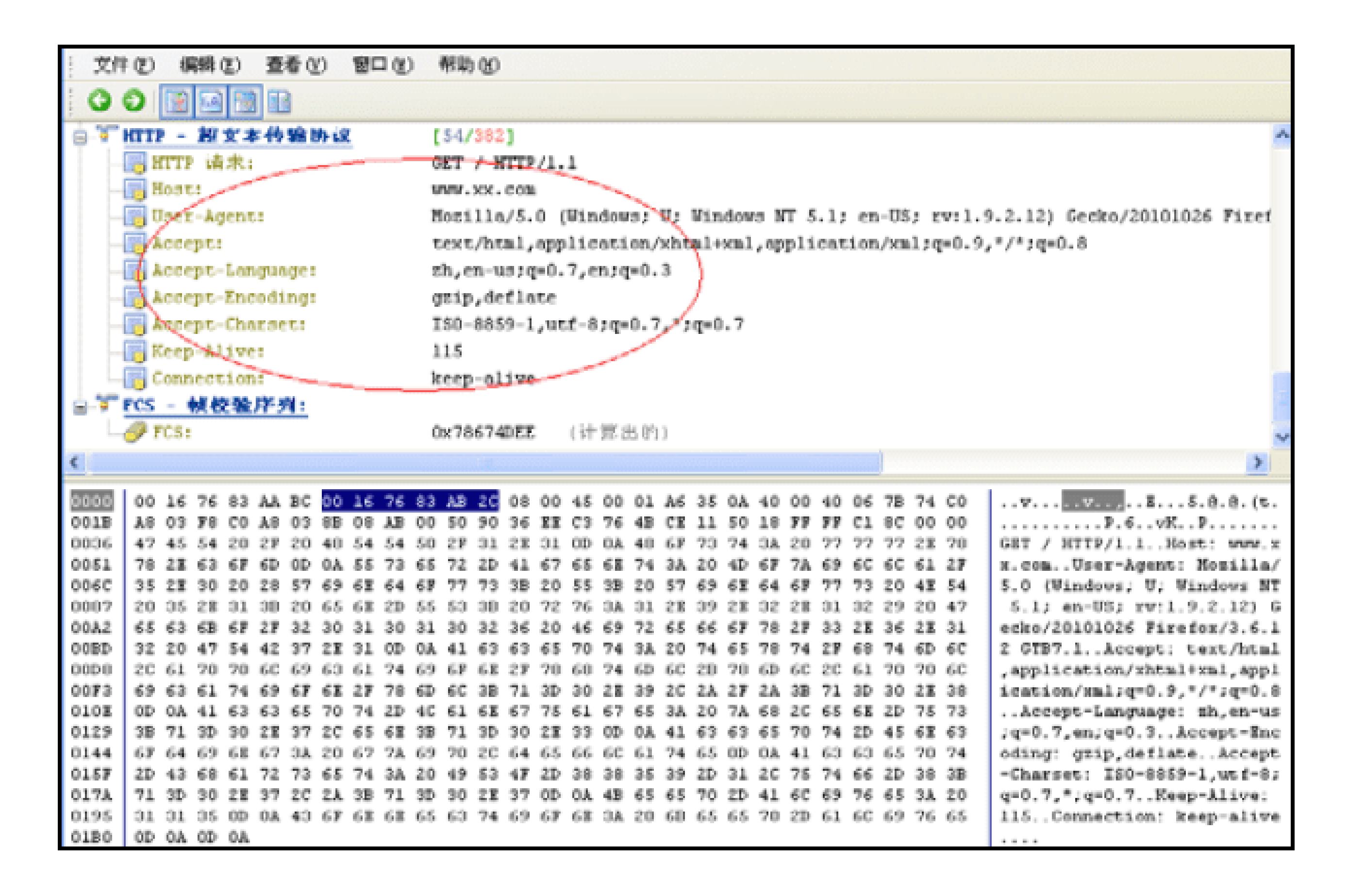
 location ~ \.flv$ {
   auth_basic "FLV Server";
   auth_basic_user_file  /usr/local/nginx-1.0.2-mp4-flv/conf/passwd1;
   root /var/www/flv;
   flv;
}
 location ~ \.mp4$ {
   auth basic "MP4 Server";
   auth basic user file  /usr/local/nginx-1.0.2-mp4-flv/conf/passwd2;
   root /var/www/mp4;
   mp4;
     }
   }
```

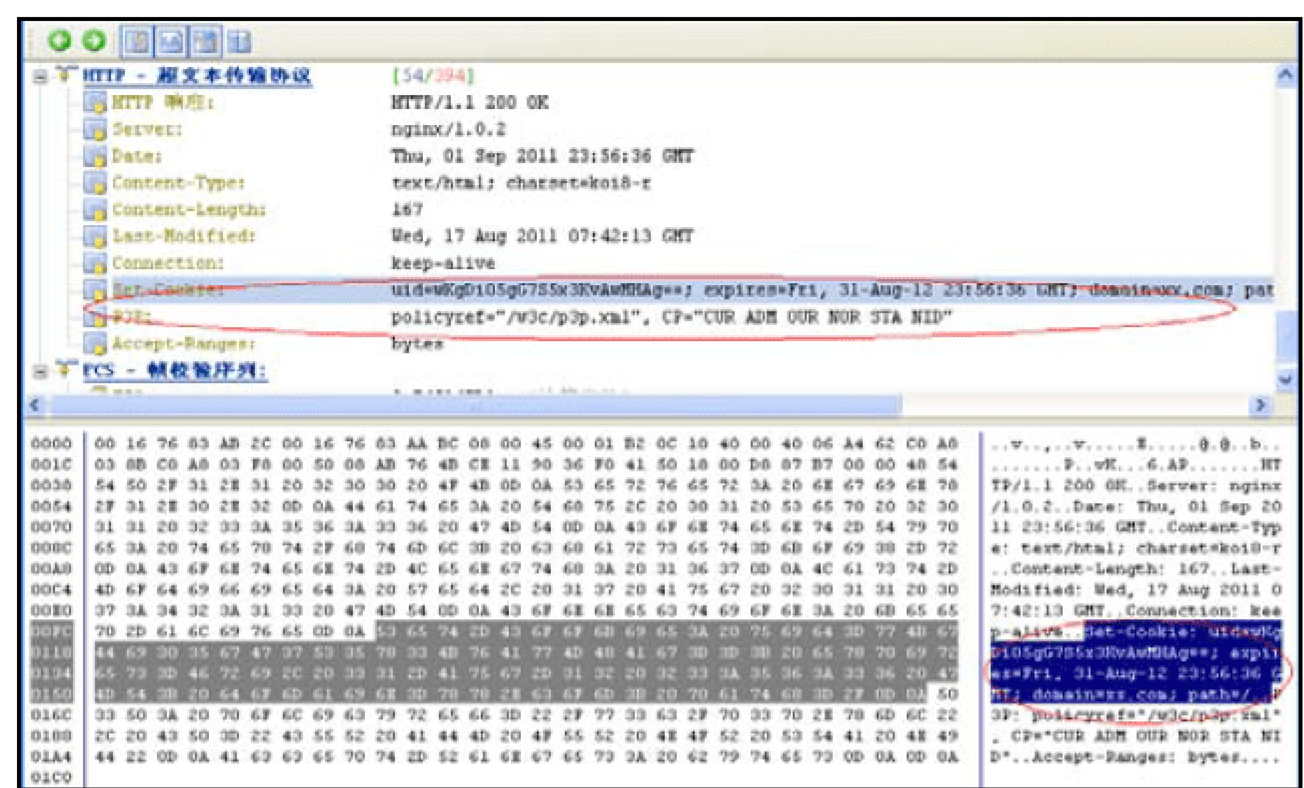
测试访问

下面我们访问 http://www.xx.com，看一下客户端发出的请求数据包。

在这里我们看到并没有：cookie 头。

服务器端发送回客户端的响应数据包：





在服务器端发送回来的数据包中存在 Set-Cookie：

```
Set-Cookie: uid=wKgDi05gG7S5x3KvAwMHAg==; expires=Fri, 31-Aug-12 23:56:36
GMT; domain=xx.com; path=/
```

这个“wKgDi05gG7S5x3KvAwMHAg==”虽简单，却不容易破解。有关 cookie 的长度，我们可以从它的源代码中找到答案：

```
if (n == NGX_DECLINED) {
return ctx;
}

ngx_log_debug1(NGX_LOG_DEBUG_HTTP, r->connection->log, 0,
"uid cookie: \"%V\"", &ctx->cookie);

if (ctx->cookie.len < 22) {
cookies = r->headers in.cookies.elts;
ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
  "client sent too short userid cookie \"%V\"",
  &cookies[n]->value);
return ctx;
```

```
}

src = ctx->cookie;

/*
 * we have to limit the encoded string to 22 characters because
 *  1) cookie may be marked by "userid_mark",
 *  2) and there are already the millions cookies with a garbage
 * instead of the correct base64 trail "=="
 */

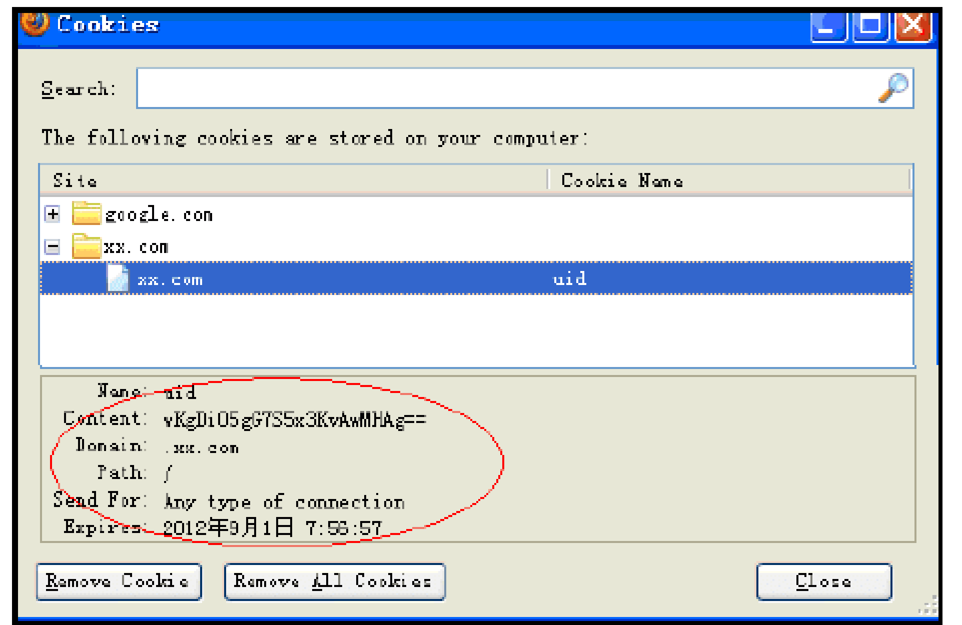
src.len = 22;

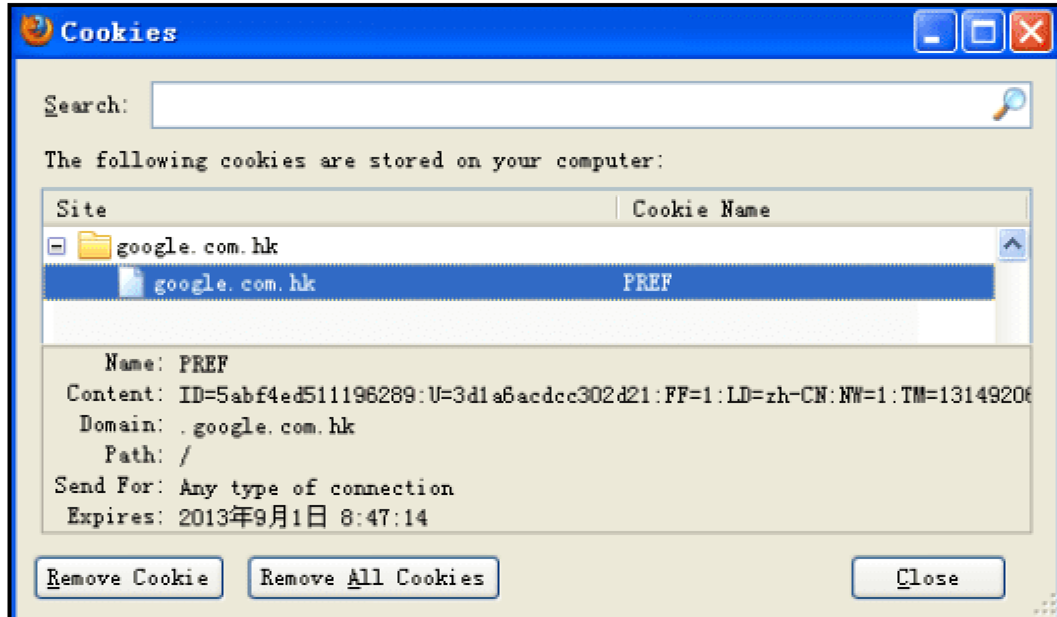
dst.data = (u_char *) ctx->uid_got;

if (ngx_decode_base64(&dst, &src) == NGX_ERROR) {
cookies = r->headers_in.cookies.elts;
ngx_log_error(NGX_LOG_ERR, r->connection->log, 0,
  "client sent invalid userid cookie \"%V\"",
  &cookies[n]->value);
return ctx;
```

该 cookie 被存储在浏览器中，以便于以后访问使用：

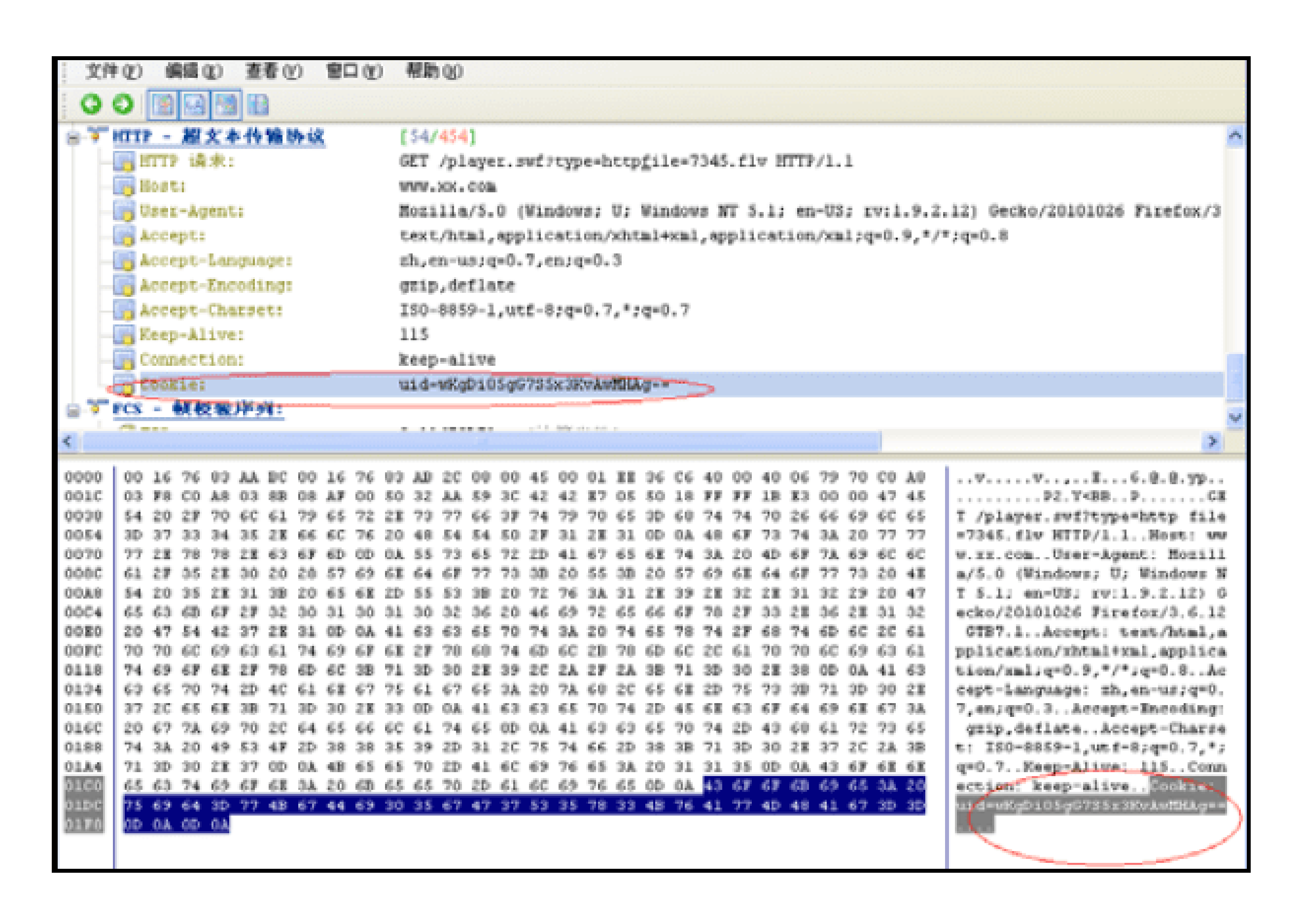
不同网站颁发的 cookie，位数有很大不同。以下是 google 颁发的 cookie：

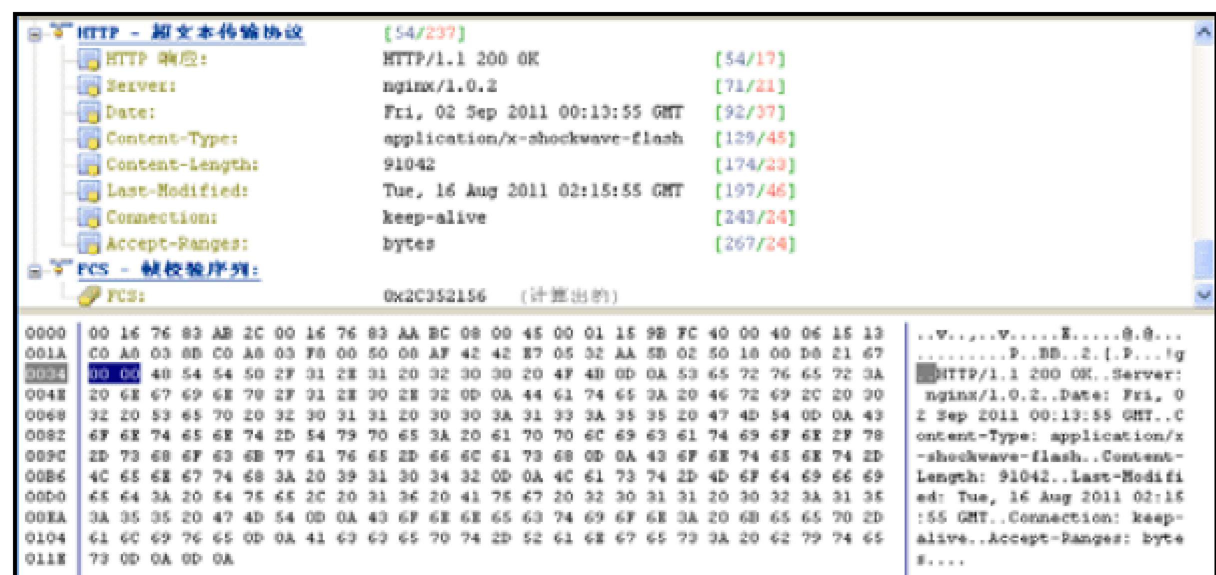




关闭浏览器后，重新打开，我们再访问该网站的其他页面，例如：http://www.xx.com/player.swf?type=http&file=7345.flv，看一下客户端发出的请求数据包：

我们看到客户端向服务器端递送了由该服务器颁发的 cookie。也正因为如此，服务器端不再向客户端颁发新的 cookie：





我们看一下 Nginx 的日志记录：

```
    [root@mail log]# tail -f custom.log

    02/Sep/2011:09:22:13
+0800|www.xx.com|561|545|-|uid=8B03A8C0C52F604E5F7DD26302030303
    02/Sep/2011:09:23:52
+0800|www.xx.com|382|545|-|uid=8B03A8C02830604E5F7DD26302040303
    02/Sep/2011:09:24:41
+0800|www.xx.com|568|545|-|uid=8B03A8C05930604E5F7DD26302050303
    02/Sep/2011:09:24:41
+0800|www.xx.com|311|719|uid=8B03A8C05930604E5F7DD26302050303|-
    02/Sep/2011:09:25:31
+0800|www.xx.com|656|157|uid=8B03A8C05930604E5F7DD26302050303|-
    02/Sep/2011:09:25:42
+0800|www.xx.com|361|157|uid=8B03A8C05930604E5F7DD26302050303|-
    02/Sep/2011:09:26:08
+0800|www.xx.com|354|157|uid=8B03A8C0C52F604E5F7DD26302030303|-
```

在这个日志中，前面的三条是颁发 cookie 的记录。后面的四条是使用 cookie 的记录。从这些 uid 的记录中虽然可以看到一些规律，但是如果不熟悉 Nginx 的代码也没法分析。因此如果你对 Nginx 的代码熟悉，特别是对该模块的源文件 ngx_http_userid_filter_module.c 部分熟悉，那么可以从那里得到答案。

第 40 章　Nginx 基于客户端请求头的访问分类

Nginx 从 0.8.37 版本开始提供一个叫做 ngx_http_split_clients_module 模块，该模块的功能从它名字很容易看出，其功能就是用基于某些条件（例如，IP 地址、头、cookie，等等）将客户端访问的资源分开。

1. 配置示例

```
http {
split_clients "${remote_addr}AAA" $variant {
0.5% .one;
2.0% .two;
* "";
}

server {
location / {
 index index${variant}.html;
```

2. 指令

split_clients 模块只提供了一条指令。

指令名称：split_clients

语法：split_clients　source_hash　$variable { ... }

默认值：none

使用环境：http

功能：该指令用于对哈希源数据计算，从 1.0.1 版本之后的 Nginx，该指令对源字符串进行计算时使用的不再是 CRC32，而改为使用 MurmurHash2 算法，然后根据哈希的百分比作为源的值。

3. 变量

split_clients 模块提供了一个变量，那就是$variant。

变量名称：$variant

功能：通过该变量的值就会判断访问者最终访问的去处。

4. 使用实例

在 Nginx 服务器的配置文件中添加以下配置内容：

```
http {
  include    mime.types;
  default_type  application/octet-stream;
```

```
    sendfile   on;

    keepalive timeout  65;

    log format  custom  $time local  |  $server name  |  $request length  |
$bytes sent | $remote addr | $variant;

    split_clients  "${remote_addr}"   $variant {
   10% .1;
   20% .2;
   30% .4; 40% .5;
    *  "";
   }

    server {
   listen   80;
   server_name  www.xx.com;

   access log logs/custom.log custom;

   location / {
     root   html;
     index index${variant}.html;
   }

   ...
   }
```

Nginx 的 Web 目录结构:

```
[root@mail html]# tree -L 1
.
|-- member
|-- simg0
|-- upload
  ...
|-- files
|-- index.html
|-- index.1.html
|-- index.2.html
|-- index.3.html
|-- index.4.html
'-- index.5.html
```

```
8 directories, 12 files
```

我们看一下 Nginx 的访问日志：

```
[root@mail logs]# tail -f  custom.log
03/Sep/2011:12:15:41 +0800|www.xx.com|321|887|100.100.170.248|.4
03/Sep/2011:12:15:46 +0800|www.xx.com|319|887|192.168.3.248|.5
03/Sep/2011:12:16:00 +0800|www.xx.com|615|389|61.135.169.106|.2
03/Sep/2011:12:18:27 +0800|www.xx.com|565|389|23.123.123.128|.2
03/Sep/2011:12:19:26 +0800|www.xx.com|563|887|192.168.1.164|.4
03/Sep/2011:12:20:09 +0800|www.xx.com|563|887|192.168.4.88|.5
03/Sep/2011:12:24:11 +0800|www.xx.com|566|887|119.184.137.242|.5
03/Sep/2011:12:25:13 +0800|www.xx.com|566|887|173.242.125.196|.5
```

第 41 章　通过 Upstream 模块使得 Nginx 实现后台服务器集群

在 Nginx 服务器中提供了一个简单的负载均衡模块，它就是 HttpUpstreamModule 模块，它的原理是基于客户端 IP 的轮询，因此，对于多台后台服务器来说是一个很好的选择。

1. 配置示例

```
upstream backend  {
  server backend1.example.com weight=5;
  server backend2.example.com:8080;
  server unix:/tmp/backend3;
}

server {
  location / {
proxy pass  http://backend;
  }
}
```

2. 指令

该模块提供了以下 3 条指令。

指令名称：ip_hash

语法：ip_hash

默认值：none

使用环境：upstream

功能：如果使用了该指令，那么将会导致客户端的请求以客户端的 IP 地址分布在 upstream 中的 server 之间。它的关键技术在于对这个请求客户端 IP 地址进行哈希计算，这种方法保证了客户端请求总是能够传递到同一台后台服务器，但是如果该服务器被认定为无效，那么这个客户端的请求将会被传递到其他服务器，因此，这种机制是一个高概率将客户端请求总是连接到同一台服务器。

另外，如果使用这个指令，那么就不能使用权重（weight）方法，如果一个在 upstream 中指定的一台 server，这个服务器需要移除（比如，需要维修），那么需要在该 IP 或者是机器名之后添加 down 参数。例如：

```
upstream backend {
  ip hash;
  server   backend1.example.com;
  server   backend2.example.com;
  server   backend3.example.com  down;
```

```
  server   backend4.example.com;
}
```

指令名称：server

语法：server name [parameters]

默认值：none

使用环境：upstream

功能：该指令用于设置服务器的 name，对于 name，可以使用域名字、IP 地址、端口或者是 UNIX 套接字，如果一个域名被解析到多个 IP 地址，那么所有的 IP 地址都将会被使用。对于 parameters，可选的 parameters 如下。

- weight = NUMBER：用于设置服务器的权重。如果没有设置，那么它将会等于 1。
- max_fails = NUMBER：该参数用于设置在一定的时间内（这个时间由 fail_timeout 参数设置）客户端对同一后台服务器可以进行尝试连接的次数，如果在这个次数之后仍然没有连接成功，那么该服务器将会被看做无效。如果没有设置该参数，那么尝试的次数将会是 1 次，如果设置为 0，那么将会关闭检测。那么什么被看做是失败，也就是判断失败的标准，是由 proxy_next_upstream 或者 fastcgi_next_upstream 的设置来决定的，因此对于客户端访问出现的 404 错误将不会导致 max_fails 的次数增加。
- fail_timeout = TIME：该参数用于设置客户端尝试连接后台服务器的时间，在这个时间内，可能将会完成由 max_fails 设置的最多次数。 如果没有设置该参数，那么默认为 10 秒。fail_timeout 和 upstream 的响应时间没关系，使用 proxy_connect_timeout 和 proxy_read_timeout 指令可以控制响应时间。
- down：如果为某一个 server 设置了该参数，那么标记了这台 server 将永久离线。通常这个参数与 ip_hash 一同使用。
- backup：该参数在 0.6.7 版本中提供，它是一个备用标志，即如果出现所有的非备份服务器都宕机或繁忙无法接受连接时，那么才会使用本服务器。需要注意的是，该参数无法和 ip_hash 指令一起使用。

例如：

```
upstream  backend  {
  server   backend1.example.comweight=5;
  server   127.0.0.1:8080  max_fails=3  fail_timeout=30s;
  server   unix:/tmp/backend3;
}
```

> 注意：如果我们在 upstream 中仅设置了一个 server，那么 Nginx 会在内部变量中设置为 1，这就意味着 max_fails 和 fail_timeout 将不会被处理。

结果：如果 Nginx 不能连接到上游服务器，那么请求将会失败。

解决：多次使用同一台服务器。例如：

```
upstream  backend  {
  server   backend1.example.comweight=5;
  server   backend1.example.commax_fails=3  fail_timeout=30s;
```

```
    server   backend1.example.com;
  }
```

指令名称：upstream

语法：upstream name { ... }

默认值：none

使用环境：http

功能：该指令是一个组合性指令，它描述了一组服务器，这组服务器将会被指令 proxy_pass 和 fastcgi_pass 作为一个单独的实体使用，它们可以将 server 监听在不同的端口，而且还可以同时使用 TCP 和 UNIX 套接字监听。服务器可以设置不同的权重，如果没有设置权重，那么默认会将其设置为 1。例如：

```
  upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080   max_fails=3  fail_timeout=30s;
    server unix:/tmp/backend3;
  }
```

对于这个配置，请求将会按照轮询的方式将其分配，当然也会根据服务器的权重分配。例如，假设我们现在有 7 个请求，那么 5 个请求将会被发送到 backend1.example.com 服务器，剩余的 2 个，“127.0.0.1:8080” 和 “unix:/tmp/backend3”各一个。当然这是在理想情况下，即我们的这些 server 都是健康活泼型的，如果有一个出现无法连接，那么就会按照顺序重新请求，直到请求完成。如果真出现这三台 server 都宕掉的情况，那么这个配置就无法提供访问了。

3. 变量

Nginx 服务器从 0.5.18 版本起，可以通过日志模块来记录变量，例如：

```
log_format timing  '$remote_addr - $remote_user [$time_local]  $request '
                    'upstream_response_time $upstream_response_time '
                    'msec $msec request_time $request_time';

log_format up_head  '$remote_addr - $remote_user [$time_local]  $request '
                 'upstream_http_content_type $upstream_http_content_type';
```

HttpUpstreamModule 模块提供了以下 5 个变量。

变量名称：$upstream_addr

功能：该变量表示了处理该请求的 upstream 中 server 的地址。

变量名称：$upstream_cache_status

功能：该变量出现在 Nginx 0.8.3 版本中，可能的值如下。

- MISS：缓存中未被命中。
- EXPIRED：生存期期满，请求被传递到后端服务器。
- UPDATING：生存期期满，陈旧的响应被使用，因为 proxy/fastcgi_cache_use_stale 升级。
- STALE：生存期期满，陈旧的响应被使用，因为 proxy/fastcgi_cache_use_stale。

- HIT：缓存命中。

变量名称：$upstream_status

功能：该变量为 upstream 中 server 的响应状态。

变量名称：$upstream_response_time

功能：upstream server 的响应时间，单位为秒，能够精确到毫秒。如果有多个 server 响应回答，那么会用逗号和冒号分隔开。

变量名称：$upstream_http_$HEADER

功能：HTTP 协议头。例如：$upstream_http_host。

4. 使用实例

在 Nginx 服务器的配置文件中添加以下内容：

```
 upstream memcached {

    server 192.168.5.18:11211;
server 192.168.5.18:11212;
server 192.168.5.19:11211;
server 192.168.5.19:11212;
   }
```

这是一个使用 Memcached 服务器的例子。

有关该模块的使用在此不再多举例了，在本书中对于它的使用有很多例子。

第 42 章　根据浏览器选择主页

根据不同的浏览器选择不同的主页，可以通过 HttpBrowserModule 来实现。该模块会创建三个变量，而变量的值取决于浏览器的请求头“User-agent”的值。根据实际的访问需要可以在配置中使用以下三个变量。

- $modern_browser：如果客户端浏览器被识别为一个现代的（modern）浏览器，那么该变量值会被指令 modern_browser_value 指定的值取代。
- $ancient_browser：如果客户端浏览器被识别为一个旧的（old）浏览器，那么该变量的值会被指令 ancient_browser_value 指定的值取代。
- $msie：如果客户端使用一个 Microsoft IE 浏览器，那么该变量被设置为 1。

为了帮助 Nginx 认识 Web 浏览器，就需要告诉它什么是现代浏览器，什么是过时的浏览器。因此，我们需要插入多条 modern_browser 和 ancient_browser 指令。

对于 HttpBrowserModule 模块提供的 modern_browser 和 ancient_browser 两个指令，可以从字面理解为“新浏览器”和“旧浏览器”。但是，对于“旧浏览器”的设定，我们可以随意一些，即更人为一些，将任何浏览器都可以设置为“旧浏览器”，但是在设置 modern_browser 指令时就不同了，其浏览器参数是预定好的，不能随便设置。在实际的使用中，可以简单地分为两类，将手机或者是其他没有被 modern_browser 指令内置指定的浏览器类型设为“旧浏览器”，而其余的设置为“新浏览器”。

例如：

```
modern_browser opera 10.0;
```

上面的这个例子中，如果用户的“User-Agent HTTP header”包含“Opera 10.0”，那么客户端浏览器就被看做是现代浏览器。

通过使用 HttpBrowserModule 模块，能够对各种浏览器的请求进行区别对待。在 Nginx 的配置中通过使用 If 指令和正则表达式的结合可以实现，与单纯的使用正则表达式相比，该模块能够提供更好的性能，而且还使得配置文件更加清晰。但是美中不足的是，它的功能较弱，只能是采用“不是……就是……”的形式。就是说如果我们想实现，IE 6、IE 7 和 Firefox 分别对应三个不同的首页，那么该模块实现不了，还得使用正则表达式来实现。

> 另外需要说明的一点是，该模块在默认安装时会被安装，如果不需要该模块，那么在安装时需要指定-without-http_browser_module 选项，取消对其安装。

1. 配置示例

根据浏览器的不同而选择 index 文件：

```
modern browser value "modern.";
modern browser msie 5.5;
modern_browser gecko 1.0.0;
modern_browser opera 9.0;
```

```
modern_browser safari 413;
modern_browser konqueror 3.0;
index index.${modern_browser}html index.html;
```

重定向旧的浏览器：

```
modern browser msie 5.0;
modern browser gecko 0.9.1;
modern_browser opera 8.0;
modern_browser safari 413;
modern_browser konqueror 3.0;
modern_browser unlisted;
ancient browser Links Lynx Netscape4;

if  ($ancient_browser) {
rewrite  ^  /ancient.html;
}
```

2. 指令

指令名称：ancient_browser

语法：ancient_browser line [line...]

默认值：no

使用环境：http，server，location

功能：当在“User-agent”字段中被识别出的浏览器为旧的浏览器时，该指令会指定出一些子链，其中，有一个特别的行“netscape4”相当于正则表达式“^Mozilla/[1-4]”。

指令名称：ancient_browser_value

语法：ancient_browser_value line

默认值：ancient_browser_value 1

使用环境：http，server，location

功能：该指令为变量 ancient_browser 指定值。

指令名称：modern_browser

语法：modern_browser browser version|unlisted

默认值：no

使用环境：http，server，location

功能：该指令指定哪一个版本的浏览器被作为现代的浏览器。当前可指定的值（浏览器）有：msie、gecko（基于 Mozilla）opera、safari、konqueror。还可以指定版本号，格式为（按大小排序）：X、X.X、X.X.X 或 X.X.X.X，它们中各自的最大值为——4000、4000.99、4000.99.99 和 4000.99.99.99。还有一个特殊值“unlisted”，它表示被考虑为现代浏览器，而没有使用指令 modern_browser 和 ancient_browser 指出的浏览器。在头中没有包括“User-agent”的浏览器被考虑为古老的浏览器（ancient），除非在“modern_browser unlisted”中指出。

指令名称：modern_browser_value

语法：modern_browser_value line

默认值：modern_browser_value 1

使用环境：http，server，location

功能：该指令为变量$modern_browser 指定一个值。

3. 举例

注意，下面提供给指令 ancient_browser 的值不连续，因此，需要按照以下格式：

```
ancient_browser "MSIE 4.0" "MSIE 5.0" "MSIE 5.5" "MSIE 6.0";
```

使用实例（来自 https://gist.github.com/228769）：

```
    #
    # Supported browsers
    #

    modern browser gecko 1.9;

    # note that Safari related directives match
    # Chrome, Mobile Safari, Palm Pre and other WebKit-based browsers here, too,
#thanks to all of them using almost identical User-Agent strings
    modern browser safari 3.0;
    modern browser safari 3.1;
    modern browser safari 3.2;
    modern_browser safari 4.0;

    modern_browser msie 7.0;
    modern_browser msie 8.0;

    modern browser  unlisted;

    #
    # Non-supported browsers
    #

    ancient browser msie 4.0;
    ancient browser msie 5.0;
    ancient browser msie 5.5;
    ancient browser msie 6.0;

    # Here is how one can disable support of a specific browser
    ancient_browser opera 7;
    ancient_browser opera 8;
    ancient_browser opera 9;
```

```
ancient_browser opera 10;

ancient_browser konqueror 3;
ancient browser konqueror 4;

ancient browser Links Lynx Netscape4;

#
# Now, to the fun part. If we perform a rewrite on every
# request here, images won't be displayed, so we have to
# do some extra work besides just
#
# if ($ancient_browser) {
#   rewrite ^ /unsupported browser.html;
#   break;
# }

set $unsupported_browser_rewrite do_not_perform;

if ($ancient_browser) {
  set $unsupported browser rewrite  perform;
}

if ($uri !~* /\.(jpeg|jpg|png|ico|gif|js|css)$/) {
  set $unsupported_browser_rewrite  do_not_perform;
}

if ($unsupported_browser_rewrite = perform) {
  rewrite ^ /unsupported browser.html;
  break;
}
```

仅支持最新版本的 Chrome、Firefox、Internet Explorer、Safari、Mobile Safari 和 Palm Pre。

又如（本例来自互联网）：

以下这个配置用于对手机浏览器的判断：

```
 modern browser  unlisted;

ancient_browser "GoBrowser";
ancient_browser "MIDP";
ancient_browser "WAP";
ancient browser "UP.Browser";
ancient browser "Smartphone";
ancient_browser "Obigo";
```

```
ancient_browser "Mobile";
ancient_browser "AU.Browser";
ancient_browser "wxd.Mms";
ancient_browser "WxdB.Browser";
ancient_browser "CLDC";
ancient_browser "UP.Link";
ancient_browser "KM.Browser";
ancient_browser "UCWEB";
ancient_browser "SEMC-Browser";
ancient_browser "Mini";
ancient_browser "Symbian";
ancient_browser "Palm";
ancient_browser "Nokia";
ancient_browser "Panasonic";
ancient_browser "MOT-";
ancient_browser "SonyEricsson";
ancient_browser "NEC-";
ancient_browser "Alcatel";
ancient_browser "Ericsson";
ancient_browser "BENQ";
ancient_browser "BenQ";
ancient_browser "Amoisonic";
ancient_browser "Amoi";
ancient_browser "Capitel";
ancient_browser "PHILIPS";
ancient_browser "SAMSUNG";
ancient_browser "Lenovo";
ancient_browser "Mitsu";
ancient_browser "Motorola";
ancient_browser "SHARP";
ancient_browser "WAPPER";
ancient_browser "LG-";
ancient_browser "LG/";
ancient_browser "EG900";
ancient_browser "CECT";
ancient_browser "Compal";
ancient_browser "kejian";
ancient_browser "Bird";
ancient_browser "BIRD";
ancient_browser "G900/V1.0";
ancient_browser "Arima";
ancient_browser "CTL";
ancient_browser "TDG";
```

```
ancient_browser "Daxian";
ancient_browser "DBTEL";
ancient_browser "Eastcom";
ancient_browser "EASTCOM";
ancient_browser "PANTECH";
ancient_browser "Dopod";
ancient_browser "Haier";
ancient_browser "HAIER";
ancient_browser "KONKA";
ancient_browser "KEJIAN";
ancient_browser "LENOVO";
ancient_browser "Soutec";
ancient_browser "SOUTEC";
ancient_browser "SAGEM";
ancient_browser "SEC";
ancient_browser "SED-";
ancient_browser "EMOL";
ancient_browser "INNO55";
ancient_browser "ZTE";
ancient_browser "iPhone";
ancient_browser "Android";
ancient_browser "Windows CE";
ancient_browser "DX";
ancient_browser "TELSON";
ancient_browser "TCL";
ancient_browser "oppo";
ancient_browser "ChangHong";
ancient_browser "MALATA";
ancient_browser "KTOUCH";
ancient_browser "TIANYU";
ancient_browser "TOUCH";
ancient_browser "MAUI";
ancient_browser "J2ME";
ancient_browser "BlackBerry";
ancient_browser "yulong";
ancient_browser "coolpad";

if ( $ancient_browser )
{
proxy_pass http://m.xxx.com;
}
```

4. 使用实例

在这个实例中，我们将 Firefox 设定为“旧浏览器”，而将 msie 6.0 设置为“新浏览器”，

这样设置纯粹是为了验证效果，而没有实际的意义。因此，不要效仿。

添加配置

```
server {
listen   80;
server_name  localhost;

location / {
root   html;
index  index.html index.htm;
}

modern_browser msie 6.0;
ancient browser Firefox;
if  ($ancient_browser) {
rewrite  ^ /ancient/index.html;

}

if  ($modern_browser) {
rewrite  ^ /modern/index.html;

}
```

目录结构及文件

以下是目录结构：

```
[root@web html]# tree
.
|-- 50x.html
|-- ancient
|   '-- index.html
|-- index.html
|-- modern
|   '-- index.html
'-- ok.html

6 directories, 14 files
```

文件内容：

```
[root@web html]# more  ancient/index.html  modern/index.html
::::::::::::::
ancient/index.html
::::::::::::::

<html>
```

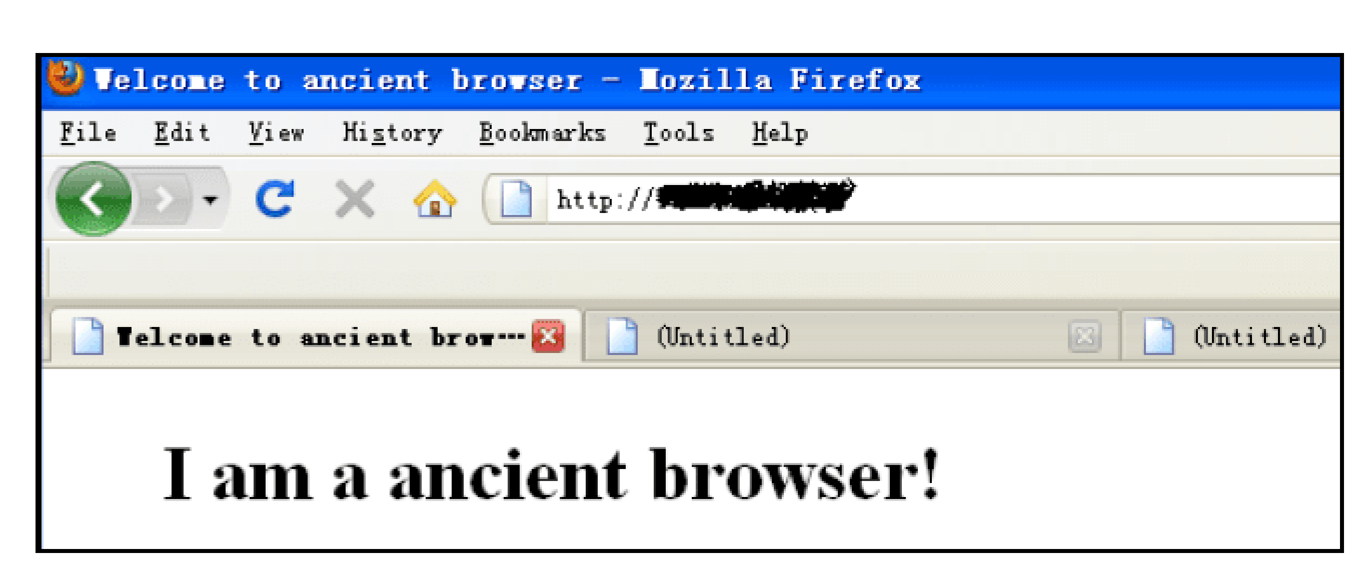
```
<head>
<title>Welcome to ancient browser</title>
</head>
<body bgcolor="white" text="black">
<center><h1>I am a ancient browser!</h1></center>
</body>
</html>
::::::::::::::
modern/index.html
::::::::::::::
<html>
<head>
<title>Welcome to modern browser!</title>
</head>
<body bgcolor="white" text="black">
<center><h1>I am a modern browser!</h1></center>
</body>
</html>
```

测试访问

用 IE 6 访问 http://www.xx.com。

用 Firefox 访问 http://www.xx.com。





第 43 章 关于 Nginx 提供下载.ipa 或.apk 文件的处理方法

1. 问题

在使用 Nginx 提供下载.ipa 或.apk 文件时会出现以下问题：通过 IE 浏览器下载会出现：替换扩展名为.zip；而使用火狐浏览器下载则会出现流的形式，就是不会下载，而是以乱码的形式浏览，显然这都不是我们想要的。

2. 解决

在网上看到一位同学的解决方法是这样的：在 conf/mime.typs 内加上：

```
 application/vnd.android.package-archive apk;
```

不知道其他的同学有没有用这个方法成功过，但这个方法在我这里是行不通的。

我的解决方法如下：

在相应的 server 下添加如下内容：

```
  location /download {
root   /sdc/ download;
  index  index.html index.htm;
  }
```

将提供被下载的.ipa 或.apk 文件放置在/sdc/ download 目录中就可以了。

第 44 章　SCGI

SCGI（Simple Common Gateway Interface）与 FastCGI 相似，也是 CGI 的一个替代协议。SCGI 源于 Python 社区。

SCGI 模块能够使得 Nginx 与 SCGI 进程互相配合工作，并且能够控制将什么参数传递到 SCGI 进程，该模块从 Nginx 服务器 0.8.42 版本开始提供使用。

1. 配置示例

```
location / {
  include scgi params;
  scgi_pass localhost:9000;
}
```

具有缓存的示例：

```
http {
  scgi cache path  /path/to/cache  levels=1:2
   keys zone=NAME:10m
   inactive=5m;

  server {
location / {
  scgi pass127.0.0.1:9000;
  scgi cache   NAME;
  scgi cache valid   200 302  1h;
  scgi_cache_valid   301  1d;
  scgi_cache_valid   any  1m;
  scgi_cache_min_uses  1;
  scgi_cache_use_stale error  timeout invalid_header http_500;
}
  }
}
```

2. 指令

SCGI 模块提供了以下指令。

指令名称：scgi_bind

语法：scgi_bind address

默认值：none

使用环境：http，server，location

功能：在调用 connect()之前，该指令会将每一个上游连接的套接字绑定到本地的 IP 地址上。

如果主机有多个网卡接口/别名，并且你想通过指定的网卡接口/IP 地址对外连接，那

么这个功能将会起作用。

例如：

```
scgi_bind  192.168.1.1;
```

指令名称：scgi_buffer_size

语法：scgi_buffer_size the_size

默认值：scgi_buffer_size 4k/8k

使用环境：http，server，location

功能：该指令用于设置缓存区的大小，缓存区用于缓存从 SCGI 服务器获取的响应的第一部分，这部分响应内容是一个简单的响应头。缓存的默认值通常来说等于由 scgi_buffers 指令设置的一个缓存区的大小。

指令名称：scgi_buffers

语法：scgi_buffers the_number is_size

默认值：scgi_buffers 8 4k/8k

使用环境：http，server，location

功能：该指令设置了缓存的大小和数量，用于缓存从 SCGI 服务器读取的数据。默认情况下，一个缓存（buffer）等于一个内存页面，具体的值依赖于具体的操作系统，可能是 4KB，8KB 或者 16KB。

指令名称：scgi_busy_buffers_size

语法：scgi_busy_buffers_size size

默认值：scgi_busy_buffers_size ["#scgi_buffer_size"] * 2

使用环境：http，server，location，if

功能：设置“忙”缓存的大小，按照一般的用法来说就是将其设置为 scgi_buffers 的两倍。

指令名称：scgi_cache

语法：scgi_cache zone|off

默认值：off

使用环境：http，server，location

功能：为缓存实际使用的共享内存指定一个区域，并定义一个 cache 区域（zone）。对区域（zone）定义标识符是为了进一步在其他指令中使用。相同的区域可以用在不同的地方。

指令名称：scgi_cache_bypass

语法：scgi_cache_bypass variable1 variable2...

默认值：none

使用环境：http，server，location

功能：该指令指定什么情况下客户端的请求将会“绕开”读取被缓存的响应，并将客户端请求传递到上游服务器。该指令从 0.8.46 版本开始提供。例如：

```
scgi cache bypass $cookie nocache  $arg nocache$arg comment;
scgi_cache_bypass $http_pragma $http_authorization;
```

如果变量为空或者是为“0”，那么表达式将会为 false。例如，在上面的这个例子中，如果

在请求中设置了“nocache”，那么这些请求总是被传递到后台服务器。

需要注意的是，来自于后台的这些响应仍旧有资格被缓存，鉴于此方式，可以通过它来刷新缓存中的某个缓存条目，具体的方式是通过发送一个具有自己选择的（请求）头的请求，例如，“My-Secret-Header: 1”，于是，便可以设置一个 scgi_no_cache 行。例如：

```
scgi_no_cache $http_my_secret_header;
```

以此方式来刷新缓存的条目。

指令名称：scgi_cache_key

语法：scgi_cache_key line

默认值：none

使用环境：http，server，location

功能：该指令用于设置缓存 key。例如：

```
scgi_cache_key localhost:9000$request_uri;
```

指令名称：scgi_cache_methods

语法：scgi_cache_methods [GET HEAD POST]

默认值：scgi_cache_methods GET HEAD

使用环境：http，server，location

功能：该指令用于设置可以缓存的 HTTP 方法，GET 和 HEAD 被默认包含在内。换句话说，它们也是不可以禁用的。如果想缓存 POST 方法，那么可以这样设置：

```
scgi_cache_methods POST;
```

指令名称：scgi_cache_min_uses

语法：scgi_cache_min_uses the_number

默认值：scgi_cache_min_uses 1

使用环境：http，server，location

功能：该指令用于设置同样的 URL 被访问多少次之后就会被缓存，默认值是访问一次之后就会被缓存。

指令名称：scgi_cache_path

语法: scgi_cache_pathpath[levels=m:n] keys_zone=name:size [inactive=time] [max_size=size]

默认值：none

使用环境：http

功能：该指令指定了缓存条目存储的位置和其他一些参数。所有的数据被存储在文件中，缓存 key 和缓存文件的名字都被经过 MD5 计算。参数 levels 设置了被用于存储缓存文件的数量和名字的宽度。例如，通过下面的指令：

```
scgi_cache_path  /data/nginx/cache  levels=1:2   keys_zone=one:10m;
```

那么数据将会被存储为以下格式：

```
/data/nginx/cache/c/29/b7f54b2df7773722d382f4809d65029c
```

即最终放置在缓存中条目的文件名称为：b7f54b2df7773722d382f4809d65029c。

缓存的数据首先被写入临时文件，然后被移动到最终的缓存目录中，从 0.8.9 版本开始，可以

将临时文件和缓存文件放置在不同的文件系统上。但是需要明白的一点是，如果这样做，那么将会替代了一个“廉价的”并且也是“原子的”通过系统调用完成文件复制的操作，因此，还是最好使用同一个文件系统，即将 scgi_temp_path 和 scgi_cache_path 的参数设置在同一文件系统中。

另外，所有活动的 key 和有关缓存数据的信息都保存在共享存储的 zone 中，这个 zone 的名字和大小都是由 key_zone 参数来指定的。假如被缓存的数据在一定的时间内没有被访问，那么可以通过指定 inactive 参数，该参数的功能就是使得在指定的这个时间段之后，如果被缓存的条目没有被访问过，那么缓存条目将会被“逐出”缓存，如果没有指定该参数的值，那么它的默认值将会被使用，默认值为 10 分钟。

使用参数 max_size 来设置可用的最大缓存，有一个特殊名称的进程叫 cache manager，该进程用于删除不活动的缓存条目和控制缓存的大小。在前面我们讲过 max_size 参数，缓存的最大值由 max_size 参数定义。当缓存中缓存的条目超过由 max_size 参数设置的最大值外，那么将会使用最近最少使用原则对数据进行清除，以便为新的缓存条目腾出空间。

指令名称：scgi_cache_use_stale

语法：scgi_cache_use_stale updating|error|timeout|invalid_header|http_500

默认值：scgi_cache_use_stale off

使用环境：http，server，location

功能：该指令用于定义 Nginx 在某些情况下（例如，网关失败、超时，无效的头，等等）是否提供过期的缓存数据。

指令名称：scgi_cache_valid

语法：scgi_cache_valid [http_return_code [...]] time

默认值：none

使用环境：http，server，location

功能：该指令用于为不同的应答设置不同的缓存时间。对于任何高速访问的站点来说，这个指令必须使用，它的依据是响应代码，因此实际上是针对响应代码来定制不同的缓存时间。例如：

```
scgi cache valid  200 302  10m;
scgi_cache_valid  404  1m;
```

在这个例子中，针对 HTTP 响应代码为 200、302 的缓存时间设置为 10 分钟，而对于 404 错误代码的缓存时间则设置为 1 分钟。

如果在 Nginx 的配置文件中进行了如下设置：

```
scgi_cache_valid  5m;
```

就是说仅指定了时间，这实际上是一种默认的写法，在这种情况下，那么代码为 200、301 和 302 的应答将会被缓存。

还有另外一种特殊的写法：

```
scgi_cache_valid  any 1m;
```

这种写法意味着将会缓存所有响应代码的条目为 1 分钟。

指令名称：scgi_connect_timeout

语法：scgi_connect_timeout time

默认值：scgi_connect_timeout 60

使用环境：http，server，location

功能：该指令用于设置与 SCGI 服务器连接的超时间隔。需要注意的是，不要将这个时间设置得超过 75 秒。

指令名称：scgi_hide_header

语法：scgi_hide_header name

使用环境：http，server，location

功能：默认情况下，Nginx 不会将“Status”和“X-Accel-...” 头从 SCGI 服务器传递到客户端，这个指令也可以隐藏其他的头。如果必须提供“Status”和“X-Accel-...”，那么可以使用 scgi_pass_header 指令来强迫它们返回给客户端。

指令名称：scgi_ignore_client_abort

语法：scgi_ignore_client_abort on|off

默认值：scgi_ignore_client_abort off

使用环境：http，server，location

功能：如果设置为 on，即使客户端放弃请求，但是 Nginx 仍会继续处理代理请求；在另外一种情况（也就是设置为 off）下，当客户端放弃请求的时候，Nginx 也会放弃它对后台服务器的请求。

指令名称：scgi_ignore_headers

语法：scgi_ignore_headers name [name...]

使用环境：http，server，location

功能：该指令用于禁止处理指定的头，这些头由 SCGI 服务器返回的响应头。可能指定的头有：“X-Accel-Redirect”、“X-Accel-Expires”、“Expires”以及“Cache-Control”。

指令名称：scgi_intercept_errors

语法：scgi_intercept_errors on|off

默认值：scgi_intercept_errors off

使用环境：http，server，location

功能：该指令用于决定是否将后台服务器的 4xx 和 5xx 错误传递到客户端。默认情况下，如果将该指令的值设置为 off，那么所有响应都将会按照后台代理服务器原有的格式发送；如果将该指令的值设置为 on，那么 Nginx 将会拦截状态代码，然后通过 error_page 指令明确处理这些代码。如果响应的状态代码不匹配 error_page 指令，那么仍然将会按照后台服务器原有的格式发送。

指令名称：scgi_max_temp_file_size

语法：scgi_max_temp_file_size 0

默认值：none

使用环境：http，server，location

功能：该指令用于指定临时文件的最大值。如果设置该指令的值为 0，那么对于 FastCGI 请求将会禁止使用临时文件，或者指定临时文件的最大值。

例如：

```
scgi_max_temp_file_size 10m;
```

指令名称：scgi_next_upstream

语法：scgi_next_upstream error|timeout|invalid_header|http_500|http_503|http_404|off

默认值：scgi_next_upstream error timeout

使用环境：http，server，location

功能：该指令定义了什么样的请求将会被传递到下一台后端服务器。

- Error：在连接服务器的过程中发生错误，即将请求传递到服务器或者是读取服务器响应头的过程中。
- timeout：在连接服务器期间发生超时，即将请求传递到服务器或者是读取服务器响应头的过程中。
- invalid_header：服务器返回为空或者是无效的响应。
- http_500：服务器返回 500 响应。
- http_503：服务器返回 503 响应。
- http_404：服务器返回 404 响应。
- off：明确禁止将请求传递到下一台服务器。

> 需要清楚的一点是：传递请求到下一台后台服务器仅发生在当什么数据也没有传递到客户端时才成为可能——就是说，如果一个 error 或是 timeout 发生在传递请求的过程中，那么是不可能再将当前发生错误的这个请求传递给另一台不同的后台服务器。

指令名称：scgi_no_cache

语法：scgi_no_cache variable [...]

默认值：none

使用环境：http，server，location

功能：该指令用于指定什么情况下的响应将不会被缓存存储，例如：

```
scgi_no_cache $cookie_nocache  $arg_nocache$arg_comment;
scgi_no_cache $http_pragma $http_authorization;
```

如果指定的变量为空字符串或者是“0”，那么将不会被缓存。例如，在上面的这个例子中，如果请求中的 cookie 设置为“nocache”，那么响应将不会被缓存。

指令名称：scgi_param

语法：scgi_param parameter value

默认值：none

使用环境：http，server，location

功能：该指令用于指定参数，这些参数将会被传递到 SCGI 服务器。参数的值可以使用字符串、变量以及它们的组合。如果没有对该指令进行设置，那么它会从它的上一层（或者叫外层，outer level）继承；如果在当前层设置了该指令，那么相对于本级别会清除任何之前的设置。例如：

```
scgi_param  SCGI 1;
scgi_param  REQUEST_URI  $request_uri;
```

参数 REQUEST_URI 的值将会决定被执行的脚本，而 QUERY_STRING 包含了请求的参数。按照 SCGI 的标准，“scgi_param SCGI 1;”必须出现在参数列表中；而“scgi_param CONTENT_LENGTH $content_length;” 作为第一个参数被自动包含在内。

如果处理 POST 请求，另外两个参数也是必须设置的：

```
scgi_param  REQUEST_METHOD   $request_method;
scgi_param  CONTENT_TYPE $content_type;
```

指令名称：scgi_pass

语法：scgi_pass scgi-server

默认值：none

使用环境：location，if in location

功能：该指令用于设置 SCGI 服务器监听的 TCP 套接字端口或者是 UNIX 套接字。例如，TCP 套接字方式：

```
scgi_pass   localhost:9000;
```

使用 UNIX 套接字的方式：

```
scgi_pass   unix:/tmp/scgi.socket;
```

也可以使用 upstream 指令设置的服务器集群：

```
upstream backend  {
  server   localhost:1234;
  server   192.168.1.3:9000
  unix:/tmp/scgi.socket;
}

scgi_pass   backend;
```

指令名称：scgi_pass_header

语法：scgi_pass_header name

使用环境：http，server，location

功能：该指令用于明确指定传递到客户端的（命名）头。

指令名称：scgi_pass_request_body

语法：scgi_pass_request_body on|off

默认值：scgi_pass_request_body on

使用环境：http，server，location

功能：该指令用于设置是否将请求体传递到 SCGI 服务器。该指令的值似乎总是应该设置为 on。

指令名称：scgi_pass_request_headers

语法：scgi_pass_request_headers on|off

默认值：scgi_pass_request_headers on

使用环境：http，server，location

功能：该指令用于设置是否将请求头传递到 SCGI 服务器。该指令的值似乎总是应该设置为 on。

指令名称：scgi_read_timeout

语法：scgi_read_timeout time

默认值：scgi_read_timeout 60

使用环境：http，server，location

功能：该指令用于设置等待上游服务器 SCGI 进程发送数据总的时间数。如果有长时间运行的应用，而且只有该应用脚本运行完成后才会有返回值，那么应该适当地设置这个长时间。如果在 Nginx 的错误日志中发现有上游服务器超时的记录，那么要适当地增加这个参数的值。

指令名称：scgi_send_timeout

语法：scgi_send_timeout time

默认值：scgi_send_timeout 60

使用环境：http，server，location

功能：该指令用于设置将请求转发到代理服务器的超时时间，单位为秒。需要注意的是，这个超时不是完成传递整个请求的超时，而是两个写操作之间的超时间隔。如果在此时间之后上游服务器没有发送新的数据，那么 Nginx 将会关闭连接。

指令名称：scgi_store_access

语法：scgi_store_access users:permissions [users:permission ...]

默认值：scgi_store_access user:rw

使用环境：http，server，location

功能：该指令用于设定创建缓存文件及目录的权限。例如：

```
scgi_store_access  user:rw  group:rw  all:r;
```

如果为 group 或者 all 设置了任何权限，那么就没必要为 user 设置了：

```
fastcgi_store_access  group:rw  all:r
```

指令名称：scgi_temp_file_write_size

语法：scgi_temp_file_write_size size

默认值：scgi_temp_file_write_size ["#scgi_buffer_size"] * 2

使用环境：http，server，location，if

功能：当设置了使用存储驱动器上的临时文件时，设置写缓存区大小，就是说写入临时文件的写缓存区。设置它的目的在于防止一个 worker 进程在传递文件时长时间地阻塞临时文件。它的大小一般设置为 2 倍的 scgi_buffer_size。

指令名称：scgi_temp_path

语法：scgi_temp_path path [level1 [level2 [level3]]]

默认值：scgi_temp_path scgi_temp

使用环境：http，server，location

功能：该指令用于设置存储临时文件的路径，用来存放从其他的服务器传送来的数据，可以指定一个三级子目录来建立哈希存储。level 的值用于指定被哈希处理的符号数。例

如下面的配置：

```
scgi_temp_path  /spool/nginx/scgi_temp 1 2;
```

产生的临时文件的命名方式类似于：

```
/spool/nginx/scgi_temp/7/45/00000123457
```

44.1 被传递给 SCGI 服务器的参数

传递给 SCGI 服务器的请求头是通过参数的格式来进行的，在应用程序和脚本从 SCGI 服务器运行时，这些参数通常以环境变量的格式被获取。例如，“User-agent”头被作为参数 HTTP_USER_AGENT 的值所传递。除了 HTTP 请求头之外，可能还会主观地传递其他参数，那么可以使用 scgi_param 来实现。

44.2 实例 1：Perl 语言的应用

1. 安装 Perl 模块

首先要安装必要的模块，有关 Perl 模块的依赖性，我们可以通过 CPAN 解决，如果不使用这个工具，我们似乎很难解决依赖问题，或者说是很繁琐（尽管我不主张同工具安装软件包，但对于 Perl 来说似乎是必不可少了）。

以下安装的模块都可以从 http://search.cpan.org/网站上下载，因此就不再写成下载路径了。

这里仅说明了必要的模块安装，其他的一定要通过 CPAN 来解决，Perl 模块的安装很麻烦。

2. 安装 SCGI 模块

```
[root@mfs2 ~]# tar -zxvf SCGI-0.6.tar.gz
[root@mfs2 ~]# cd SCGI-0.6
[root@mfs2 SCGI-0.6]# perl Makefile.PL
[root@mfs2 SCGI-0.6]# make
[root@mfs2 SCGI-0.6]# make install
/usr/bin/perl Build --makefile_env_macros 1 install
Building SCGI
Installing /usr/lib/perl5/site perl/5.8.5/SCGI.pm
Installing /usr/lib/perl5/site perl/5.8.5/SCGI/Request.pm
Installing /usr/share/man/man3/SCGI::Request.3pm
Installing /usr/share/man/man3/SCGI.3pm
```

该模块是执行 SCGI 与应用程序服务器的接口。

3. 安装 Plack 模块

```
[root@mfs2 ~]# tar -zxvf Plack-0.9982.tar.gz
[root@mfs2 ~]# cd Plack-0.9982
[root@mfs2 Plack-0.9982]# perl Makefile.PL
```

```
…
cp scripts/plackup blib/script/plackup
/usr/bin/perl  "-Iinc"  -MExtUtils::MY  -e  'MY->fixin ( shift ) '  --
blib/script/plackup

…
[root@mfs2 Plack-0.9982]# make
[root@mfs2 Plack-0.9982]# make install
```

Plack：Perl 的 Web 框架和 Web 服务器（PSGI 工具包）。

4. 安装 Plack-Handler-SCGI 模块

```
[root@mail ~]# tar -zxvf Plack-Handler-SCGI-0.02.tar.gz
[root@mail ~]# cd Plack-Handler-SCGI-0.02
[root@mfs2 Plack-Handler-SCGI-0.02]# perl Makefile.PL
Writing Makefile for Plack::Handler::SCGI
Writing MYMETA.yml and MYMETA.json
[root@mfs2 Plack-Handler-SCGI-0.02]# make
[root@mfs2 Plack-Handler-SCGI-0.02]# make install
Installing /usr/lib/perl5/site perl/5.8.5/Plack/Handler/SCGI.pm
Installing /usr/share/man/man3/Plack::Handler::SCGI.3pm
Appending installation info to
/usr/lib/perl5/5.8.5/i386-linux-thread-multi/perllocal.pod
```

Plack::Handler::SCGI 是 SCGI 使用的独立运行的守护进程。

5. 认识命令

plackup 是一个命令行工具，通过 Plack 服务器运行 PSGI 应用程序。它实际上也是一个 Perl 脚本，我们看一下它的内容：

```
[root@mfs3 Plack-0.9982]# vi  scripts/plackup

#!perl
use strict;
use Plack::Runner;

my $runner = Plack::Runner->new;
$runner->parse_options (@ARGV) ;
$runner->run;

__END__
```

plackup 会自动地解决它运行的环境，并且会在这个环境中运行我们的应用程序。FastCGI、CGI、AnyEvent 以及其他的模块都会被检测到，权威的列表可以参考 Plack::Loader。

plackup 会假设我们在当前的目录下有一个 app.psgi 脚本，app.psgi 最后的语句应该是 PSGI 应用程序的一个代码应用：

```
#!/usr/bin/perl
use MyApp;
my $application = MyApp->new;
my $app = sub { $application->run_psgi (@_) };
```

6. 参数

参数名称：.psgi

说明：第一个非选项参数将被作为.psgi 文件的路径。

也可以通过-a 或者--app 来设置这个路径。如果省略，那么在当前目录下的 app.psgi 将会被使用。

例如：

```
plackup --host 127.0.0.1 --port 9090 /path/to/app.psgi
```

7. 选项

选项名称：-a，--app

功能：指定.psgi 脚本的全路径，可以交替使用该选项，作为 plackup 的第一个参数。

选项名称：-e

功能：对于给定的 Perl 代码将作为 PSGI 程序进行评估。该选项类似于 Perl 的-e 选项，格式为：

```
plackup -e 'sub { my $env = shift; return […] }'
```

例如：

```
[root@mfs3www]#plackup-e'my$app=sub{my$env=shift;return[200,['Content-Type'=>'text/plain'],["Hello!Welcomefrom$env->{REMOTE ADDR}!"],];};'
HTTP::Server::PSGI: Accepting connections at http://0:5000/
```

当我们想运行自定义的应用程序时，例如，Plack::App::*，也很方便，例如：

```
plackup -MPlack::App::File -e 'Plack::App::File->new (…) ->to_app'
```

我们也可以在命令行中通过-e 选项和一个指定路径.psgi 文件来封装一个中间件配置，在-e 内的代码，也可以使用 Plack::Builder DSL 语法。例如：

```
plackup -e 'enable "Auth::Basic", authenticator => ...;' myapp.psgi
```

等于 PSGI 应用程序：

```
use Plack::Builder;
use Plack::Util;

builder {
   enable "Auth::Basic", authenticator => …;
   Plack::Util::load_psgi ("myapp.psgi") ;
};
```

> 注意，当使用-e 选项启用中间件时，plackup 不会预设定 app.psgi 路径，因此，需要明确在命令行传递它的路径，或者是在-e 中，且在 enable 之后载入应用程序。

看下面命令的执行情况：

```
plackup# 运行 app.psgi
plackup -e 'enable "Foo"'                 # 不会工作!
plackup -e 'enable "Foo"' app.psgi        # 工作
plackup -e 'enable "Foo"; sub { ... }' # 工作
```

选项名称：-o，--host

功能：该选项用于指定一个 TCP 接口。默认没有定义，就是允许服务器绑定所有的接口。该选项仅对使用 TCP 套接字连接的服务器有效。

选项名称：-p，--port

功能：该选项用于设置绑定的 TCP 端口，默认值为 5000，该选项仅对使用 TCP 套接字连接的服务器有效。

选项名称：-s，--server，“PLACK_SERVER”环境变量

功能：明确指定执行的 server，当明确提供了 server 之后，“-s”或“--server”将会变为首选，而不是环境变量。如果没有指定该选项，那么 plackup 将会基于环境变量来选择一个最好的 server 来执行。详细请参考 Plack::Loader。

选项名称：-S，--socket

功能：通过 UNIX 套接字监听连接，该选项用于设置套接字的路径。如果没有设置，该选项仅支持 UNIX 套接字的监听方式。

选项名称：-l，--listen

功能：设置监听一个或者多个地址，可以是“HOST:PORT”，“:PORT”或者“PATH”（没有冒号）。该选项可以使用多次，来监听多个地址，但是由服务器决定它是否支持多个接口。

选项名称：-D，--daemonize

功能：该选项用于设置将进程运行在后台。

选项名称：-I

功能：指定包含的 Perl 库路径，类似于 Perl 的-I 选项。可以多次使用该选项来添加多个 Perl 库。

选项名称：-M

功能：在 app 代码之前载入（命名）模块，可以多次使用该选项来添加多个模块。

选项名称：-E，--env，“PLACK_ENV”环境变量

功能：该选项用于设定环境变量。通过设置“-E”或“--env”的值也能够写到“PLACK_ENV”环境变量。这将允许应用程序或者是框架告诉环境变量设置运行。例如：

```
# 这两个的结果相同
plackup -E deployment
env PLACK_ENV=deployment plackup
```

通用的值有 development、deployment 和 test。默认值是 development，会载入中间件

AccessLog、StackTrace 和 Lint。

选项名称：-r，--reload

功能：该选项用于使得 plackup 能够在无论所部署的环境中哪一个文件发生了变化，那么服务器都能被 plackup 重启。该选项默认监控的目录是 lib 和存放.psgi 文件的基础目录（base）。使用“-R”将可以监控其他的目录。

重新载入将会延时应用程序的编译，如果为了使用的模块需要 plackup 扫描应用程序，那么自动服务器检测（参考“-s”）可能不会按照你期望的行为来实现。当使用“-r”或者“-R”的时候，要避免使用“-s”明确指定服务器。

选项名称：-R，--Reload

功能：该选项能够使得在给定的目录中文件发生变化后都将会使得 plackup 重新启动服务器。使用“-R”或“--Reload”选项时如果指定多个值，那么需要使用逗号将列表分开。例如：

```
plackup -R /path/to/project/lib,/path/to/project/templates
```

选项名称：-L，--loader

功能：该指令用于载入如何运行服务器的子类。可以设置的有 Plack::Loader（默认）、Restarter（当设置了“-r”或者“-R”选项后将会自动启用）、Delayed 和 Shotgun。更详细的可以参考 Plack::Loader::Delayed 和 Plack::Loader::Shotgun。

选项名称：--access-log

功能：该选项用于指定访问日志文件的路径。默认情况下（在开发环境中），访问日志被输出到标准的错误输出设备。

8．使用举例

例 1：从 app.psgi 文件读取应用程序。

格式：plackup

例如：

```
[root@mfs3 www]# plackup
HTTP::Server::PSGI: Accepting connections at http://0:5000/
```

如果没有指定具体的.psgi，那么默认将会去查找 app.psgi 文件。

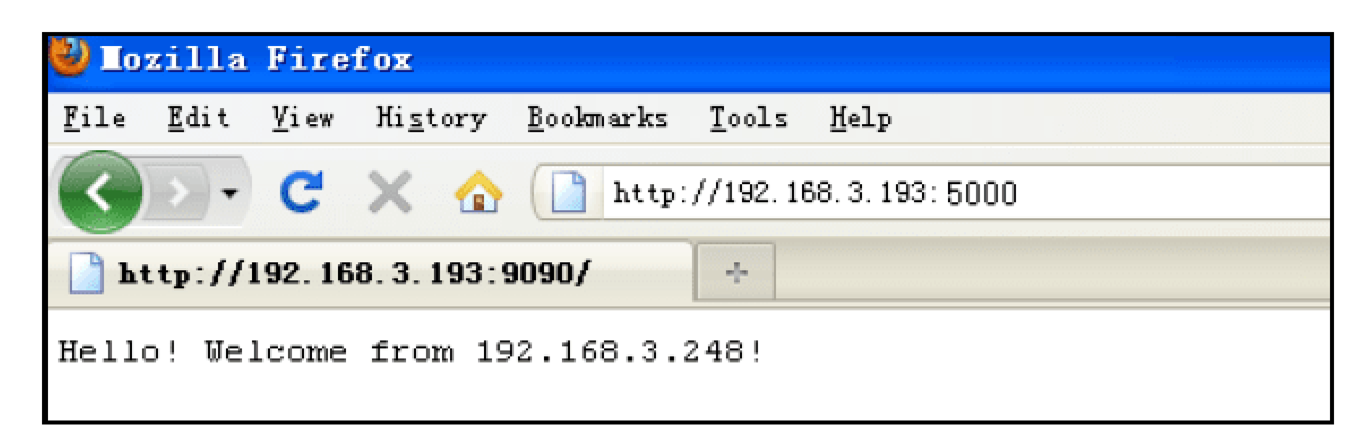
例 2：从 ARGV[0]中选择 .psgi 文件（或者使用-a 选项）。

格式：plackup hello.psgi

例如：

```
[root@mfs3 www]# plackup 4.psgi
HTTP::Server::PSGI: Accepting connections at http://0:5000/
```

可以通过 http://192.168.3.193:5000/的方式直接访问。例如：



看一下访问日志：

```
192.168.3.248 - - [19/Sep/2011:12:19:54 +0800] "GET / HTTP/1.1" 200 51 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026
Firefox/3.6.12 GTB7.1"
```

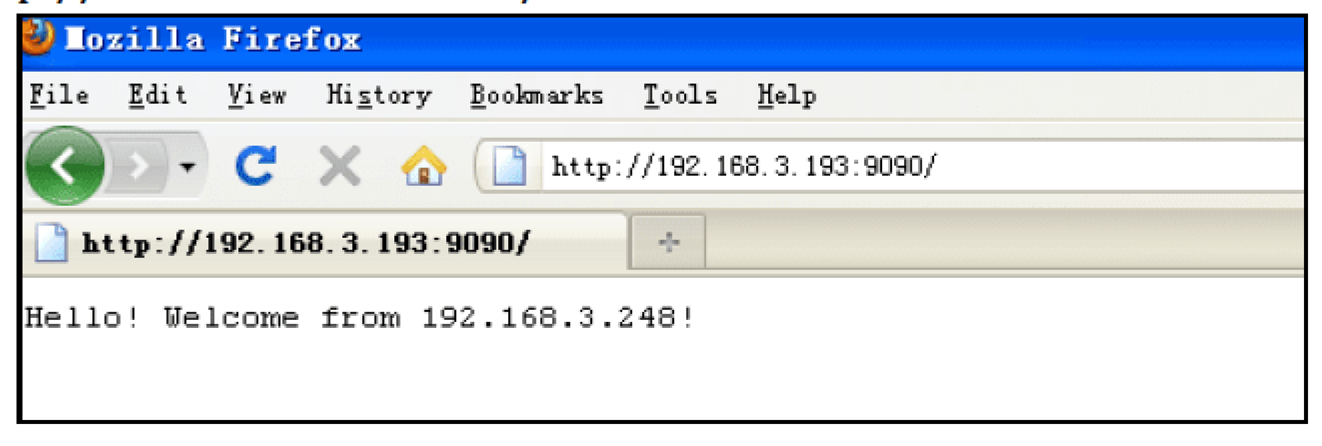
例 3：使用参数--server（或-s）切换服务器。

格式：plackup --server HTTP::Server::Simple --port 9090 --host 127.0.0.1 test.psgi

例如：

```
[root@mfs3 www]#  plackup --server HTTP::Server::Simple --port 9090 --host
192.168.3.193  4.pm
HTTP::Server::Simple::PSGI:Acceptingconnectionsathttp://192.168.3.193:90
90/
```

可以通过 http://192.168.3.193:9090/的方式直接访问，例如：



看一下访问日志：

```
192.168.3.248 - - [19/Sep/2011:12:23:41 +0800] "GET / HTTP/1.1" 200 51 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026
Firefox/3.6.12 GTB7.1"
```

例 4：使用 UNIX 套接字来运行 FCGI 守护进程。

格式：plackup -s FCGI --listen /tmp/fcgi.sock myapp.psgi

例如：

```
[root@mfs3 www]# plackup -s FCGI --listen /tmp/fcgi.sock app.psgi
FastCGI: manager (pid 30781) : initialized
FastCGI: server (pid 30782) : initialized
FastCGI: manager (pid 30781) : server (pid 30782) started
```

例 5：将 FCGI 监听在端口 9090。

格式：plackup -s FCGI --port 9090

例如：

```
[root@mfs3 www]# plackup -s FCGI --port 9090
FastCGI: manager (pid 30583) : initialized
FastCGI: server (pid 30584) : initialized
FastCGI: manager (pid 30583) : server (pid 30584) started
```

不可以通过 http://192.168.3.193:9090/ 直接访问。

需要说明的是，在 Nginx 的配置文件中将会是以下配置内容：

```
location / {
include fastcgi.conf;
fastcgi_pass 192.168.3.193:9090;
}
```

例 6： SCGI 方式。

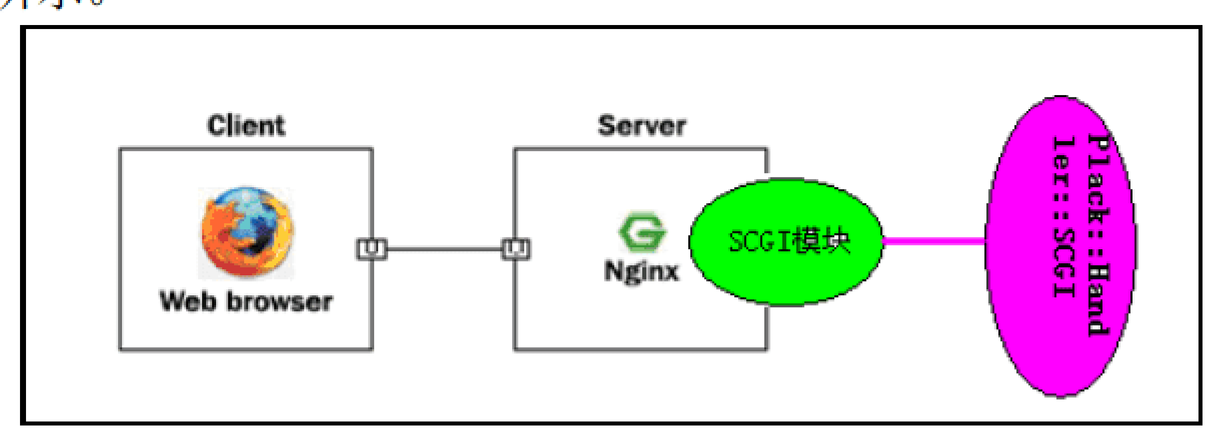
格式：plackup -s SCGI --port 22222

例如：

```
[root@mfs3 www]# plackup -s SCGI --app /var/www/4.pm --port 22222
Plack::Handler::SCGI: Accepting connections at http://0:22222/
```

9. 使用配置

架构部署如下图所示。



这也是我们要使用的方式，启动 Plack::Handler::SCGI 服务器：

```
[root@mfs3 www]# plackup -s SCGI --app /var/www/4.pm --port 22222
Plack::Handler::SCGI: Accepting connections at http://0:22222/
```

添加 Nginx 的配置文件为：

```
http {

  include   mime.types;
  default_type  application/octet-stream;


  sendfileon;
  keepalive timeout  65;

  scgi_cache_path   /tmp/scgi_cache  levels=1:2
   keys_zone=NAME:1000m
   inactive=5m;
```

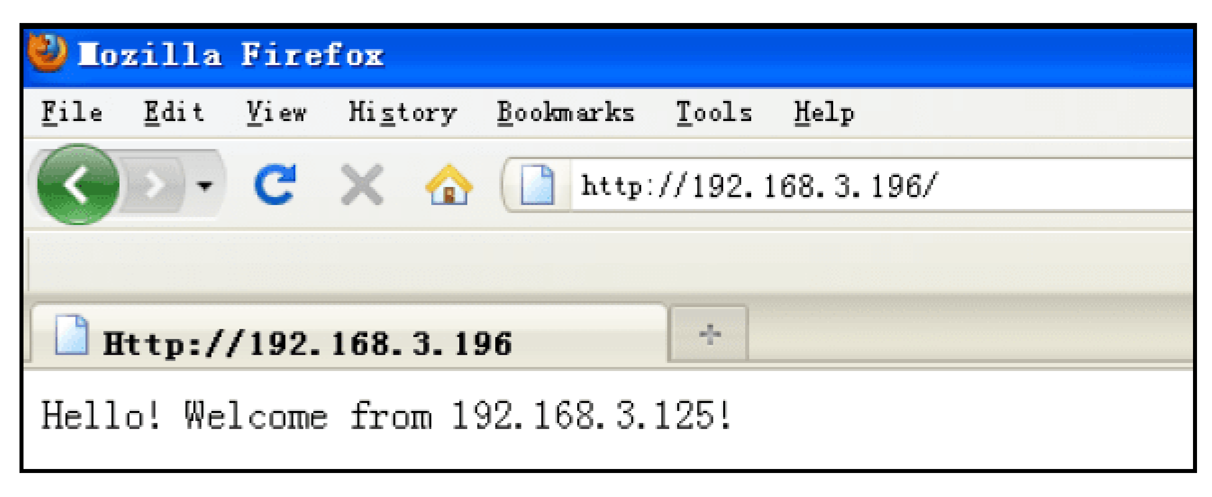
```
    server {

    listen   80;
    server name  localhost;

   location / {

    include scgi_params;
    scgi_pass192.168.3.193:22222;
    scgi_cache   NAME;
    scgi_cache_valid   200 302  1h;
    scgi cache valid   301  1d;
    scgi cache valid   any  1m;
    scgi cache min uses  1;
    scgi cache use stale error  timeout invalid header http 500;
   }
    }
   }
```

访问测试：



我们看一下 Nginx 的访问日志：

```
   [root@web36 logs]# tail -f access.log

   192.168.3.125 - - [20/Sep/2011:13:56:12 +0800] "GET / HTTP/1.1" 200 34 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026
Firefox/3.6.12 GTB7.1"
```

后台服务器的日志：

```
   192.168.3.125 - - [20/Sep/2011:13:56:12 +0800]  "GET / HTTP/1.1" 200 34 "-"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026
Firefox/3.6.12 GTB7.1"
```

缓存情况：

```
   [root@mail scgi cache]# pwd
   /tmp/scgi cache
   [root@mail scgi cache]# tree
   .
   '-- e
   '-- 27
```

```
'-- d41d8cd98f00b204e9800998ecf8427e

2 directories, 1 file
[root@mail scgi cache]# more e/27/d41d8cd98f00b204e9800998ecf8427e
◀=xNxN
KEY:
Status: 200
Content-Type: text/plain

Hello! Welcome from 192.168.3.125!
```

44.3 实例 2：Python 语言的应用

Python 的 SCGI 模块安装，其官方地址：http://python.ca/scgi/。

在 Python 语言下，要想使用 SCGI 协议，需要使用 SCGI 模块，安装了该模块后，既可以和 django 框架结合也可以和 quixote 框架结合。但是我们在这里只讲述了它们的结合，而没有深入它们的应用。quixote 框架是一个老牌的框架，但用户不是很多，它和 Apache 结合得更多；而对于 django 框架，我们使用的更多的是 uwsgi 协议，关于这一点我们在“Nginx 与 Python”中有详细的讲解。

1. 下载并安装 SCGI

```
[root@mail ~]# wget http://python.ca/scgi/releases/scgi-1.14.tar.gz
[root@mail ~]#tar -zxvf scgi-1.14.tar.gz
[root@mail ~]#cd scgi-1.14
[root@mail scgi-1.14]# python setup.py install
running install
running build
running build py
creating build
creating build/lib.linux-i686-2.7
creating build/lib.linux-i686-2.7/scgi
copying scgi/scgi_server.py -> build/lib.linux-i686-2.7/scgi
copying scgi/__init__.py -> build/lib.linux-i686-2.7/scgi
copying scgi/quixote_handler.py -> build/lib.linux-i686-2.7/scgi
copying scgi/test passfd.py -> build/lib.linux-i686-2.7/scgi
running build ext
building 'scgi.passfd' extension
creating build/temp.linux-i686-2.7
creating build/temp.linux-i686-2.7/scgi
gcc -pthread -fno-strict-aliasing -g -O2 -DNDEBUG -g -fwrapv -O3 -Wall
-Wstrict-prototypes -fPIC -I/usr/local/include/python2.7 -c scgi/passfd.c -o
build/temp.linux-i686-2.7/scgi/passfd.o
```

```
gcc-pthread-sharedbuild/temp.linux-i686-2.7/scgi/passfd.o-obuild/lib.linux-i686-2.7/scgi/passfd.so
running install_lib
creating /usr/local/lib/python2.7/site-packages/scgi
copyingbuild/lib.linux-i686-2.7/scgi/scgi server.py->/usr/local/lib/python2.7/site-packages/scgi
copyingbuild/lib.linux-i686-2.7/scgi/  init  .py->/usr/local/lib/python2.7/site-packages/scgi
copyingbuild/lib.linux-i686-2.7/scgi/quixote_handler.py->/usr/local/lib/python2.7/site-packages/scgi
copyingbuild/lib.linux-i686-2.7/scgi/passfd.so->/usr/local/lib/python2.7/site-packages/scgi
copyingbuild/lib.linux-i686-2.7/scgi/test passfd.py->/usr/local/lib/python2.7/site-packages/scgi
byte-compiling /usr/local/lib/python2.7/site-packages/scgi/scgi server.py to scgi_server.pyc
byte-compiling /usr/local/lib/python2.7/site-packages/scgi/__init__.py to __init__.pyc
byte-compiling
/usr/local/lib/python2.7/site-packages/scgi/quixote handler.pytoquixote handler.pyc
byte-compiling /usr/local/lib/python2.7/site-packages/scgi/test passfd.py to test_passfd.pyc
running install_egg_info
Writing /usr/local/lib/python2.7/site-packages/scgi-1.14-py2.7.egg-info
```

2. SCGI 的应用

关于 SCGI 的应用，我们通过以下三个实例来讲述。

SCGI 应用程序

```
[root@mail SCGI_Python]# cat scgi_server.py
#!/usr/bin/env python

import scgi;
import scgi.scgi server;

class TestHandler (scgi.scgi_server.SCGIHandler) :
def produce (self, env, bodysize, input, output) :
output.write ("Content-Type: text/plain\r\n\r\n") ;
output.write ("Hello world !!!\r\n") ;

if   name   == "  main  ":
server = scgi.scgi_server.SCGIServer (
```

```
handler_class=TestHandler,
port=9090）;
server.serve（）;
```

运行该应用：

```
[root@mail SCGI_Python]# ./scgi_server.py
```

添加 Nginx 配置文件：

```
http {

  include   mime.types;
  default type  application/octet-stream;


  sendfileon;
  keepalive_timeout  65;

  scgi cache path   /tmp/scgi cache  levels=1:2
   keys_zone=NAME:1000m
   inactive=5m;

  server {

  listen   80;
  server_name  localhost;

location / {

  include scgi params;
  scgi pass192.168.3.193:22222;
  scgi_cache   NAME;
  scgi cache valid   200 302  1h;
  scgi cache valid   301  1d;
  scgi_cache_valid   any  1m;
  scgi cache min uses  1;
  scgi_cache_use_stale error  timeout invalid_header http_500;
}
  }
}
```

访问测试：

```
[root@lin8 server]# curl   http://192.168.3.8
Hello world !!!
```

查看缓存情况：

```
[root@mail scgi_cache]# tree
.
'-- e
'-- 27
'-- d41d8cd98f00b204e9800998ecf8427e


2 directories, 1 file
[root@mail scgi_cache]# more e/27/d41d8cd98f00b204e9800998ecf8427e
0?yNyN
```

```
KEY:
Content-Type: text/plain

Hello world !!!
```

使用 quixote 框架

使用 quixote 的网站不是很多，豆瓣使用的是这个框架。需要说明的是，quixote 与 Nginx 通过 SCGI 访问衔接不是很好，在 URL 中会有问题，在这里仅给出一个使用例子，如果要想实际应用，那么可能会有很多问题在等待着你的处理。

另外，如果使用 2.7 版本有问题，那么可以使用 1.3 版本。

安装 quixote 框架

```
[root@g8 ~]# wget http://quixote.ca/releases/Quixote-2.7.tar.gz
[root@g8 ~]# tar -zxvf Quixote-2.7.tar.gz
[root@g8 ~]# python setup.py  install
```

运行 server/simple_server.py 简单测试 quixote：

```
[root@mail quixote]# server/simple_server.py
```

通过以下命令访问：

```
[root@mail ~]# lynx http://127.0.0.1:8080
```

```
                                                                    Quixote Demo
                                  Hello, world!

To understand what's going on here, be sure to read the doc/demo.txt file included with Quixote.

Here are some features of this demo:
  * simple: A Python function that generates a very simple document.
  * plain: A Python function that generates a plain text document.
  * error: A Python function that raises an exception.
  * publish_error: A Python function that raises a PublishError exception. This exception will be caught and formatted by
    the Publisher.format_publish_error() method.
  * dumpreq: Print out the contents of the HTTPRequest object.
  * css: The stylesheet for this document.
  * extras/: Demos of some of Quixote's more advanced features.
```

```
[root@mail quixote]# server/simple_server.py
debug message from the index page
localhost.localdomain - - [21/Sep/2011 16:42:27] "GET / HTTP/1.0" 200 -
```

目录 server 下还有其他的一些用法，在此就不多介绍了，毕竟我们不是讲解 quixote 框架：

```
[root@mail quixote]# tree server/
server/
|-- __init__.py
|-- _fcgi.py
|-- cgi_server.py
|-- fastcgi server.py
|-- medusa server.py
|-- mod python handler.py
|-- scgi_server.py
|-- simple_server.py
|-- twisted_server.py
```

```
'-- util.py

0 directories, 10 files
```

编写 server-scgi.py 文件，启动 SCGI 服务器：

```
[root@g8 ~]# vi server-scgi.py
#!/usr/bin/env python

from scgi.quixote_handler import QuixoteHandler, main
from quixote.publish import Publisher

from quixote import enable ptl
enable_ptl ()

class MyAppHandler (QuixoteHandler) :

publisher_class = Publisher
root_namespace = 'quixote.app
prefix = ""

if   name   == '  main  ':
main (MyAppHandler)
```

server-scgi.py 的参数：

```
[root@mfsmaster ~]# ./server-scgi.py -h
option -h not recognized
Usage: ./server-scgi.py [options]

-F -- stay in foreground (don't fork)    //只运行在前台
-P -- PID filename                       //设定 PID 文件名称
-l -- log filename                       //设定日志文件名称
-m -- max children                       //设定最大的子进程数量
-p -- TCP port to listen on              //设定监听端口号
-u -- user id to run under               //设定子进程运行用户
```

编辑模块

要测试 quixote 框架，就得编写模块。下面的模块包含三个文件（该模块来自于互联网，出处不明确）。

注意目录结构，该模块共三个文件：

```
[root@g8 ~]# pwd
/usr/local/lib/python2.4/site-packages/quixote/app
[root@mfsmaster app]# tree
.
|--   init   .py
'-- hello
```

```
|-- __init__.py
'-- hello_ui.ptl

1 directory, 3 files
[root@g8 app]# more   init   .py  hello/   init   .py hello/hello ui.ptl
::::::::::::::
  init   .py
::::::::::::::
_q_exports = ['hello']
::::::::::::::
hello/__init__.py
::::::::::::::
from app.hello.hello ui import say hello

 q exports = []

def _q_index(request):
 return say_hello("everyone")

def _q_lookup(request, name):
 def hello(request):
  return say_hello(name)
  return hello
::::::::::::::
hello/hello_ui.ptl
::::::::::::::
def say hello [html]  (name):
 header(title="Hello")
 "Hello, "
 """<em class="name">%s</em>!""" % name
 footer()

def header [html]  (title):

 """<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 <html xmlns="http://www.w3.org/1999/xhtml">
 <head>
 <title>%s</title>
 </head>
 <body>
 """ % title
```

```
def footer [html] () :
 """</body>
 </html>
```

server-scgi.py 文件的位置：

```
[root@g8 SCGI Python]# pwd
/root/SCGI_Python
[root@mfsmaster SCGI_Python]# tree
.
|-- app
|   |--   init  .py
|   '-- hello
|   |--   init  .py
|   '-- hello ui.ptl
'-- server-scgi.py

2 directories, 4 files
```

除了 server-scgi.py 文件外，其与“/usr/local/lib/python2.4/site-packages/quixote/app”中的结构和内容完全相同。

运行 server-scgi.py：

```
[root@g8 SCGI_Python]# ./server-scgi.py -p 3000 -l /var/log/scgi.log
```

该命令使得 SCGI 服务器运行在 TCP 套接字，监听端口 3000，将日志保存在/var/log/scgi.log 下。

监控 SCGI 日志：

```
[root@g8 ~]#/var/log/scgi.log
[2011-09-21 14:37:51] MyAppHandler created
```

添加 Nginx 服务器的配置：

```
  server {

  listen   80;
  server_name  localhost;

 location / {

  include scgi params;
  scgi param   SCRIPT NAME $request uri;
  scgi pass127.0.0.1:3000;
 }
  }
```

访问测试：

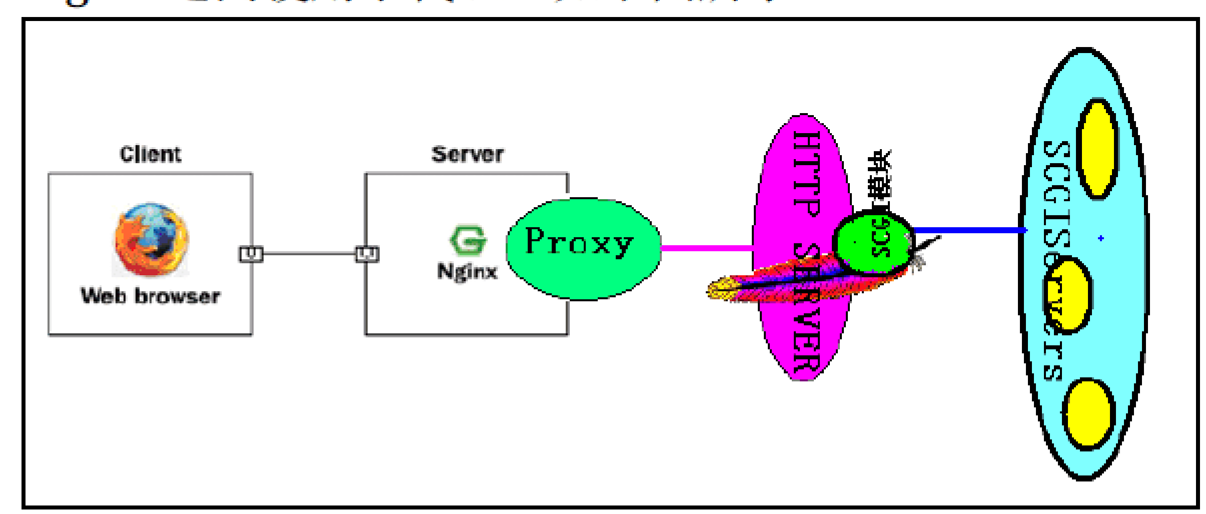
```
[root@mail ~]# lynx http://192.168.3.194/hello/
```

```
    Hello, everyone!
```

```
    192.168.3.139 - - [21/Sep/2011:14:56:54 +0800] "GET /hello/ HTTP/1.0" 200
260 "-" "Lynx/2.8.7rel.2 libwww-FM/2.14"
```

附例 Apache

这个例子是为了解决 quixote 框架与 Nginx 服务器搭配的问题，我们使用了 Apache 和 quixote 框架，在 Apache 和 Nginx 之间使用了代理，如下图所示：



在 scgi-1.14 安装包中提供了 Apache1 和 Apache2 的安装模块，在这里我们以 Apache2 为例。

使用 apxs 工具添加模块。这也是 Apache 的一个亮点功能，它不用重新编译安装 Apache，只需要载入一个模块就可以了。在这里我们使用 RH 自带的 Apache。如果使用自定义位置的安装，那么要注意该工具的位置，不要用错。

执行 apxs 命令。要注意一下它的安装位置，特别是使用非标准位置的安装：

```
    [root@mfsmaster scgi-1.14]# cd apache2
    [root@mfsmaster apache2]# ls
    Makefile  mod scgi.c  README.txt
    [root@mfsmaster apache2]# apxs -i -c mod_scgi.c
    /bin/sh /usr/lib/apr/build/libtool --silent --mode=compile gcc -prefer-pic
-O2 -g -pipe -m32 -march=i386 -mtune=pentium4 -DAP_HAVE_DESIGNATED_INITIALIZER
-DLINUX=2-D_REENTRANT-D_GNU_SOURCE-pthread-I/usr/include/apr-0-I/usr/include
/httpd  -c -o mod_scgi.lo mod_scgi.c && touch mod_scgi.slo
    /bin/sh /usr/lib/apr/build/libtool --silent --mode=link gcc -o mod scgi.la
-rpath /usr/lib/httpd/modules -module -avoid-versionmod scgi.lo
    /usr/lib/httpd/build/instdso.shSH LIBTOOL='/bin/sh/usr/lib/apr/build/lib
tool' mod scgi.la /usr/lib/httpd/modules
    /bin/sh/usr/lib/apr/build/libtool--mode=installcpmod_scgi.la/usr/lib/htt
pd/modules/
    cp .libs/mod_scgi.so /usr/lib/httpd/modules/mod_scgi.so
    cp .libs/mod scgi.lai /usr/lib/httpd/modules/mod scgi.la
    cp .libs/mod scgi.a /usr/lib/httpd/modules/mod scgi.a
    ranlib /usr/lib/httpd/modules/mod scgi.a
    chmod 644 /usr/lib/httpd/modules/mod scgi.a
    PATH="$PATH:/sbin" ldconfig -n /usr/lib/httpd/modules
    ----------------------------------------------------------------------
    Libraries have been installed in:
```

```
   /usr/lib/httpd/modules

 If you ever happen to want to link against installed libraries
 in a given directory, LIBDIR, you must either use libtool, and
 specify the full pathname of the library, or use the '-LLIBDIR'
 flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD LIBRARY PATH' environment variable
  during execution
   - add LIBDIR to the 'LD_RUN_PATH' environment variable
  during linking
   - use the '-Wl,--rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to '/etc/ld.so.conf'

 See any operating system documentation about shared libraries for
 more information, such as the ld (1)  and ld.so (8)  manual pages.
 ----------------------------------------------------------------------
 chmod 755 /usr/lib/httpd/modules/mod_scgi.so
```

为 Apache 添加配置。添加配置有两点，一是载入 SCGI 模块，二是添加访问目录：

```
LoadModule scgi modulemodules/mod scgi.so

<Location "/">
SCGIServer 127.0.0.1:3001
SCGIHandler On
</Location>
```

运行 SCGI 服务器：

```
[root@g8 SCGI_Python]# ./server-scgi.py -p 3000 -l /var/log/scgi.log
```

添加 Nginx 配置：

```
  server {

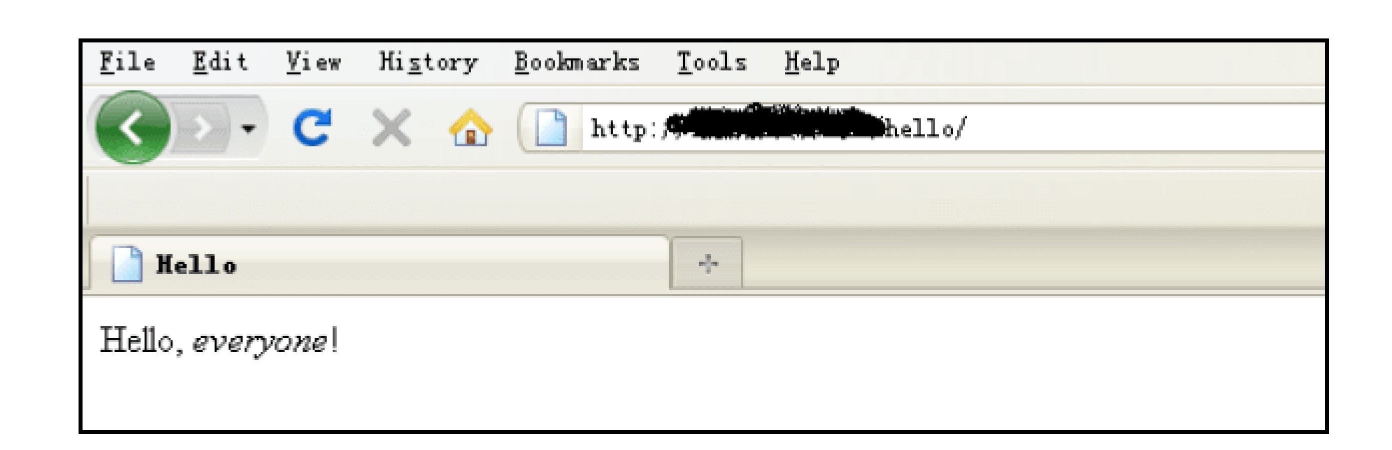
  listen   80;
  server_name  www.xx.com;

location / {

  include scgi params;
  scgi_pass192.168.15.56:3001;
  scgi_cache   NAME;
  scgi_cache_valid   200 302  1h;
  scgi_cache_valid   301  1d;
  scgi cache valid   any  1m;
  scgi_cache_min_uses  1;
```

```
    scgi_cache_use_stale error  timeout invalid_header http_500;
  }
    }
```

访问测试：



44.4 在 Nginx 中使用 Etag

在 Nginx 中使用 Etag 头分为两种情况，一种是对静态文件实现 Etag，另一种是为动态应用程序实现 Etag。我们将分别通过 nginx-static-etags 模块和 nginx-dynamic-etags 模块来实现，请看下一章。

第 45 章　Expires 与 ETag

在 Nginx 中没有使用 Etag 头，安装 Nginx 的创始人 Igor Sysoev 的观点"对于静态资源而已，看不出 Etag 比 Last-Modified 好"，因此在 Nginx 中就没有用 Etag。但是 Etag 也有自己的独到之处，它可以解决 Last-Modified 无法解决的问题——Etag。

按照业界的分析有三种情况。

下面是文件属性的状态参数：

```
[root@master ~]# stat README
  File: 'README'
  Size: 2075Blocks: 16 IO Block: 4096   regular file
Device: fd00h/64768dInode: 15466845Links: 1
Access: (0644/-rw-r--r--)  Uid: (0/root)   Gid: (0/root)
Access: 2011-07-28 21:05:00.000000000 +0800
Modify: 2011-07-28 21:05:00.000000000 +0800
Change: 2011-11-11 11:27:18.000000000 +0800
```

- 对于修改非常频繁的文件。可能会在 1 秒内修改多次，由于 Last-Modified 是基于秒级检测的，说到底是由于它检测的依据是被访问文件修改 Linux 系统的时间戳，因此在秒内的修改 Last-Modified 是无法检测出的。对于这种情况，我认为就没有必要启用缓存了。
- 对于周期性改变的文件。这种情况很常见，通常文件内容并没有发生改变而只是文件的 Modify 的改变，如果想测试这种情况，可以使用 touch 命令来 touch 一个已经存在的文件来测试。文件的内容并未改变，但是依据 Last-Modified 的原理需要重新获取。
- 有些无法精确到文件的最后修改时间。这种情况不是很多。

这两个 Http 头都是用于有效控制客户端缓存的，在 Nginx 中可以通过以下配置指令来实现：

```
location / {
  root   html;
  index  index.html index.htm;
  expires 10d;
  FileETag on;
}
```

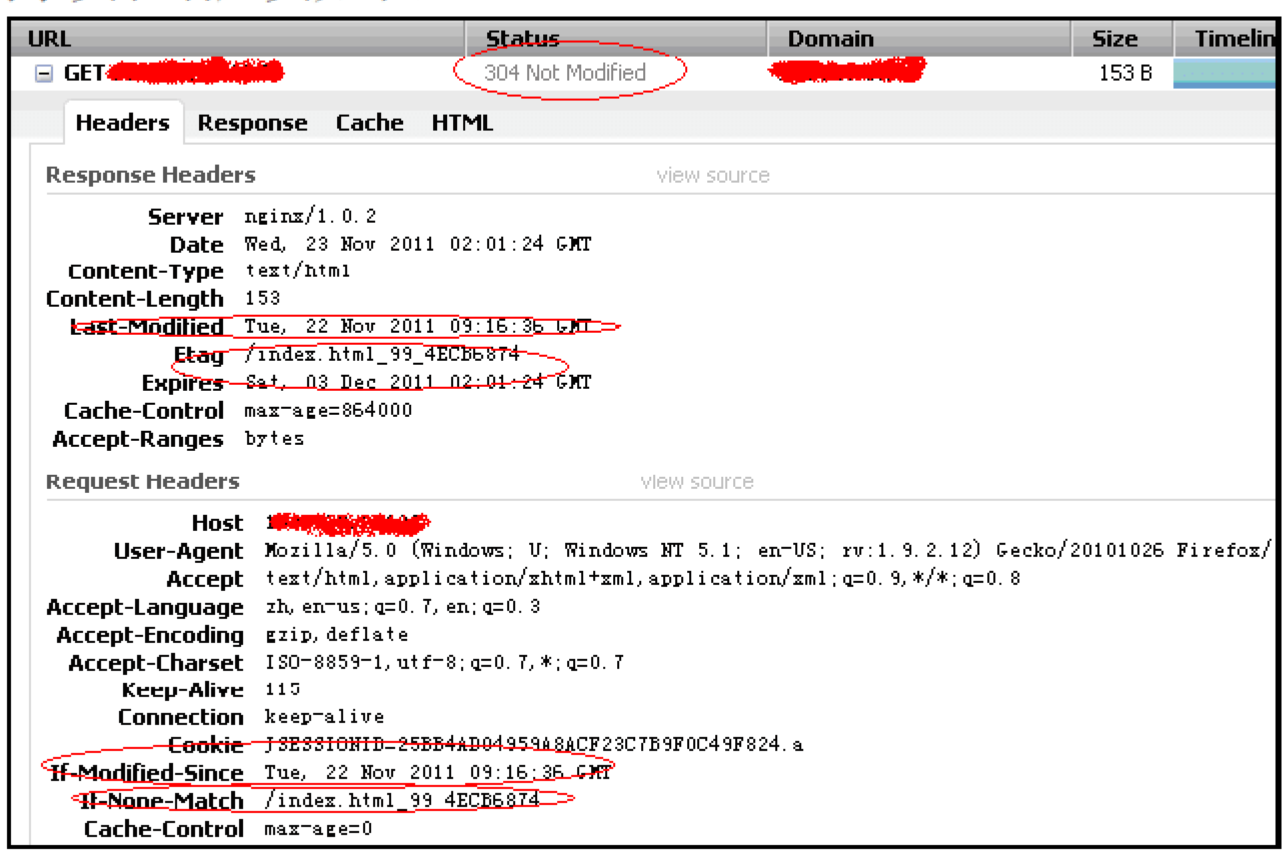
第一次访问该目录下的网页时响应头如下：

Response Headers　view source

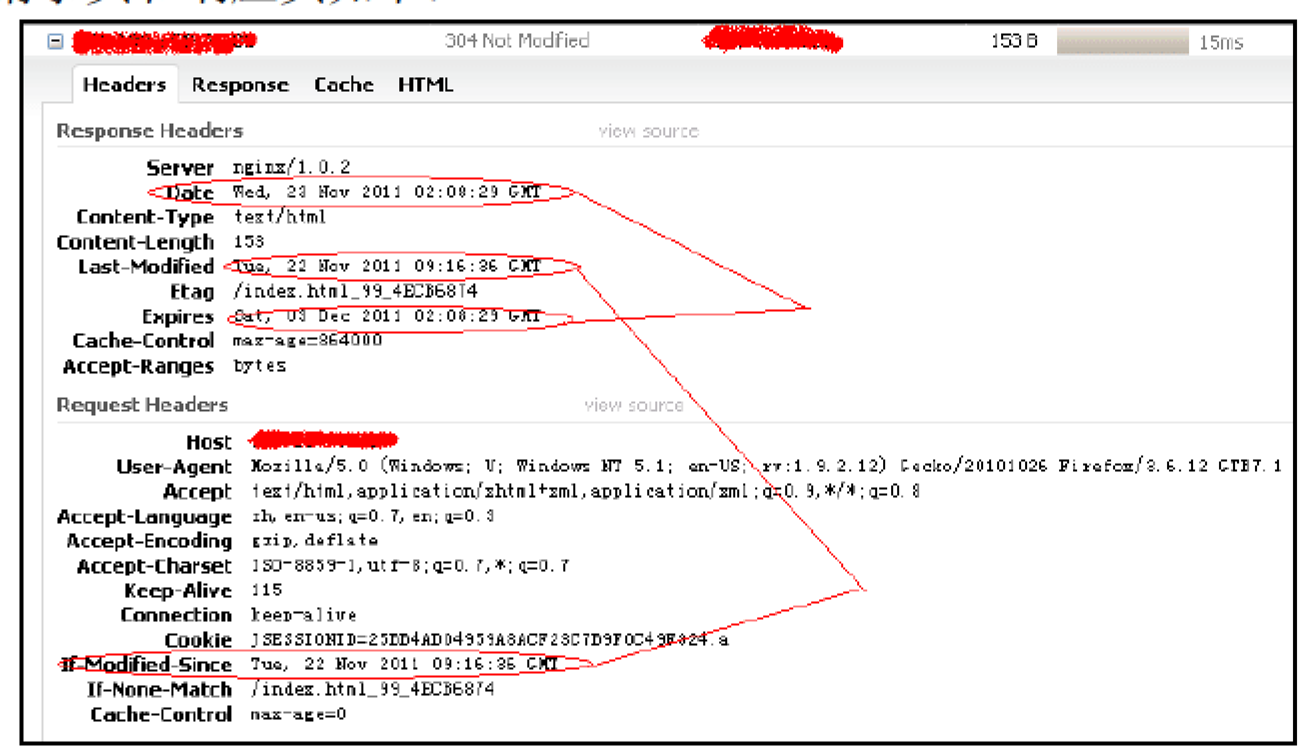
Server	nginx/1.0.2
Date	Wed, 23 Nov 2011 01:34:58 GMT
Content-Type	text/html
Content-Length	153
Last-Modified	Tue, 22 Nov 2011 09:16:36 GMT
Connection	keep-alive
Etag	/index.html_99_4ECB6874
Expires	Sat, 03 Dec 2011 01:34:58 GMT
Cache-Control	max-age=864000
Accept-Ranges	bytes

Request Headers　view source

再次访问的请求头和响应头如下：



最后访问的请求头和响应头如下：



分析这三次访问。以 Expires 为缓存的方式是使用了时间缓存，即按照 RFC2616 对 HTTP 协议的规定，在客户端第二次向服务器发出请求时，对于第一次访问请求的资源，如果响应状态为 200，那么在这次请求中将会添加一个新的请求头：If-Modified-Since，顾名思义，就是询问服务器从这个时间起，或者说是以这个时间为分割点，在这时间点之前有没有修改过这个文档，如果没有修改，那么发回的 http 状态代码是 304，并且同时再次发回响应头 Last-Modified。注意这两个头的时间完全相同。

这就是 Expires 的原理。

而 Etag 的工作原理则与 Expires 完全不同。按照 HTTP 协议的规定，对于 Etag 的值没有过规定，而只是规定了要将 Etag 的值放置在一对引号（“”）之内。因此在开发上就灵活了很多。在后面会讲到 Nginx 的第三方模块 nginx-static-etags 模块来实现对静态文件提供 Etag。

它的原理就是通过检查一对引号（“”）之内的值，或者是说由引号之内定义的值（就是说由变量生成的值，例如 URI、文件大小等，这由具体的开发来决定，如果你对开发感兴趣，那么可以查看 nginx-static-etags 的源代码，另外，如果需要自己设置 If-None-Match 匹配的值，就是说在 nginx-static-etags 模块中由 etag_format 指令产生的格式，那么也可以使用 Nginx 提供的变量来实现）。

Etag 的检测是使用了 If-None-Match 头，在客户端第一次访问一个页面时，在服务器的响应

头中会发送回一个叫做 Etag 的响应头，在前面的截图中我们也看到过，在第二次对同一个页面发出请求时在请求头中会有一个 If-None-Match 头，如果与服务器端再次生成的 Etag 匹配，那么表示请求的页面并没有改变，而是以 304 代码响应，同时在响应头中再次发回 If-None-Match 头。

这里需要说明一点，使用了 Etag 是有代价的（无论多少总是有的），它有个产生 Etag 和比较 Etag 的过程，因此会占用 CPU 资源。

> 最后一点需要说的是，在我们同时使用了 Expires 和 Etag 之后，没有谁优先的问题，而是满足两者才会做出决定。

45.1 安装 nginx-static-etags 模块

下载第三方的 nginx-static-etags 模块：

```
[root@mail ~]# wget https://nodeload.github.com/mikewest/   \
> nginx-static-etags/tarball/master
--15:13:09--
https://nodeload.github.com/mikewest/nginx-static-etags/tarball/master
Resolving nodeload.github.com... 207.97.227.252
Connecting to nodeload.github.com|207.97.227.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2685 (2.6K) [application/octet-stream]
Saving to: `mikewest-nginx-static-etags-25bfaf9.tar.gz'

100%[==============================>] 2,685   --.-K/s   in 0s

15:13:15 (488 MB/s) - `mikewest-nginx-static-etags-25bfaf9.tar.gz' saved
[2685/2685]
```

编译并安装：

```
[root@mail ~]# tar -zxvf mikewest-nginx-static-etags-25bfaf9.tar.gz
[root@mailnginx-1.0.2]#./configure--prefix=/usr/local/nginx-1.0.2-static
etags  \
> --add-module=/root/mikewest-nginx-static-etags-25bfaf9
[root@mail nginx-1.0.2]# make
[root@mail nginx-1.0.2]# make install
```

1. 配置示例

```
location / {
...
FileETag on;
...
}
```

2. 指令

nginx-static-etags 模块提供了以下两条指令。

指令名称：FileETag

默认值：off

使用环境：http，server，location

功能：该指令用于设置是否使用静态文件的 Etag 功能。该指令的可选值有两个，即 on 和 off。

指令名称：etag_format

使用环境：http，server，location

功能：用于设置 Etag 的格式，没有过多的说明。

以下是这两条指令在源代码中的定义：

```
    static ngx command t  ngx http static etags commands[] = {
    { ngx_string ( "FileETag" ) ,
      NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF FLAG,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof ( ngx_http_static_etags_loc_conf_t, FileETag ) ,
      NULL },

    { ngx_string ( "etag_format" ) ,

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof ( ngx_http_static_etags_loc_conf_t, etag_format ) ,
      NULL },

      ngx null command
    };
```

3. 使用实例

在相应的 location 中添加 FileETag 指令：

```
    location /html {
    root   /sdc;
    index  index.html index.htm;
    FileETag on;
    }


    location /pic {
    root   /sdc/;
    FileETag on;
```

```
    }

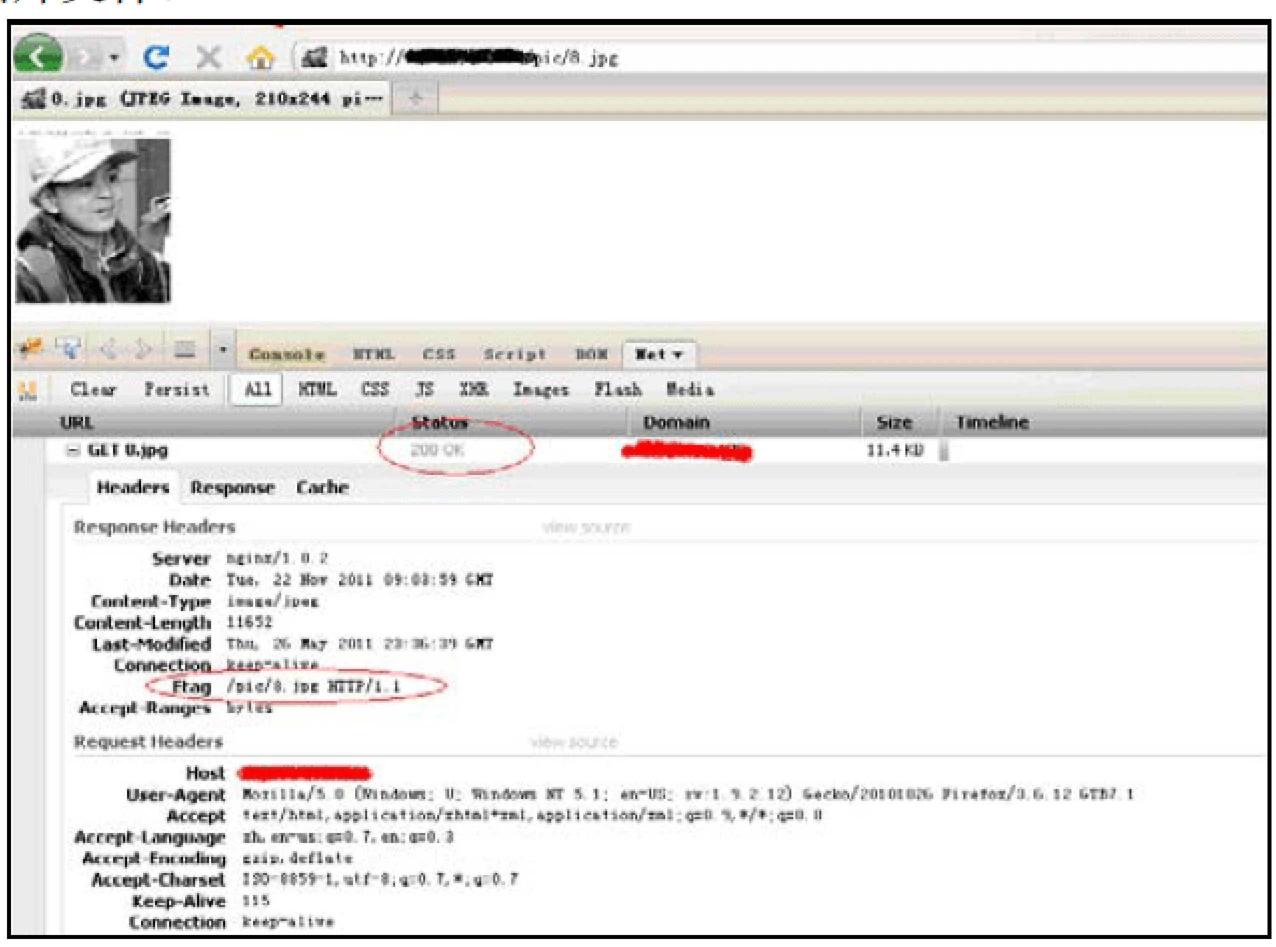
    location /css {
    root   /sdc/;
    FileETag on;
    etag format "mtl 12";
    }
```

4. 测试访问

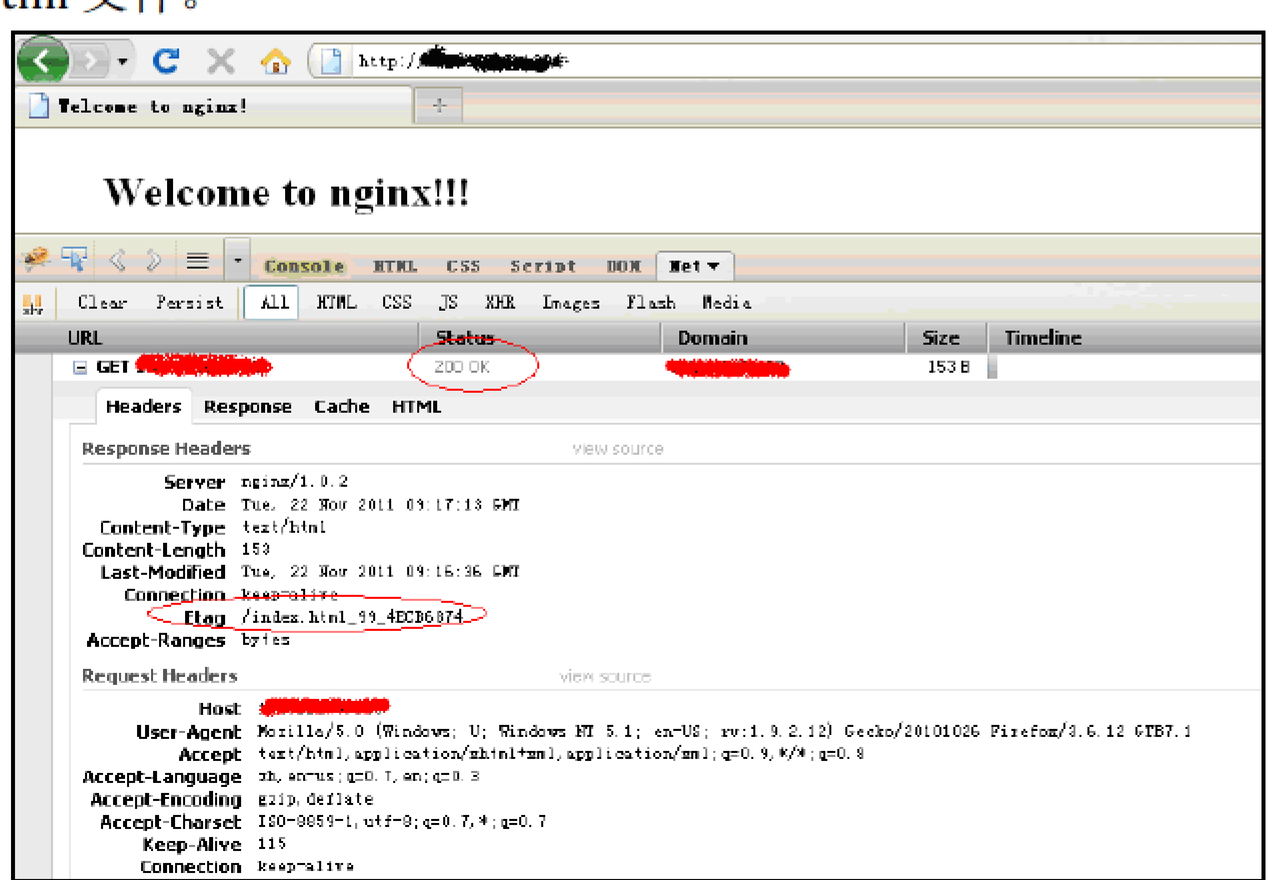
下面的测试分为两种情况：一种情况是在使用 Etag 之后第一次访问，另一种情况是在使用 Etag 之后的再次访问。

第一次访问

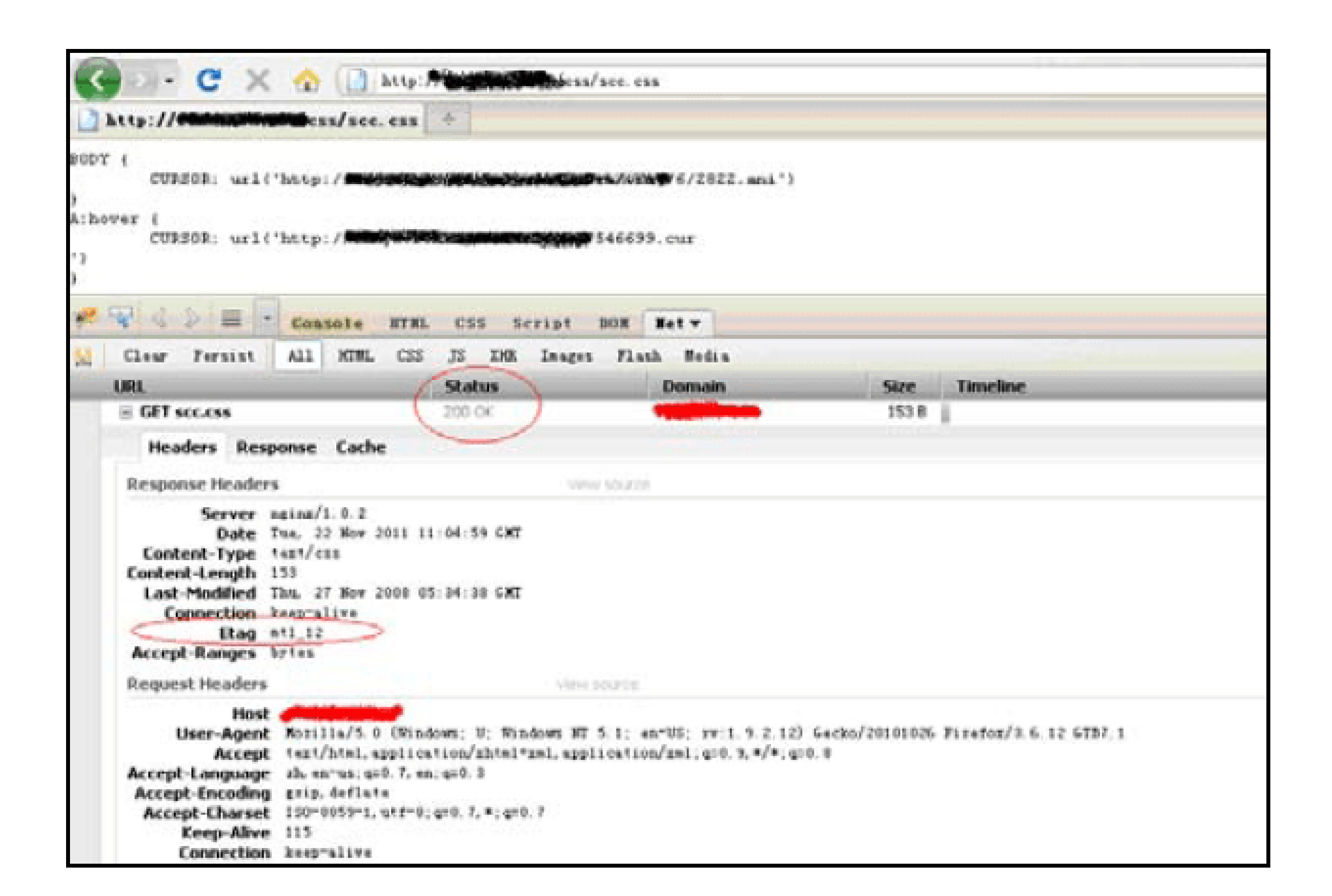
例 1：访问图片文件。



例 2：访问 html 文件。

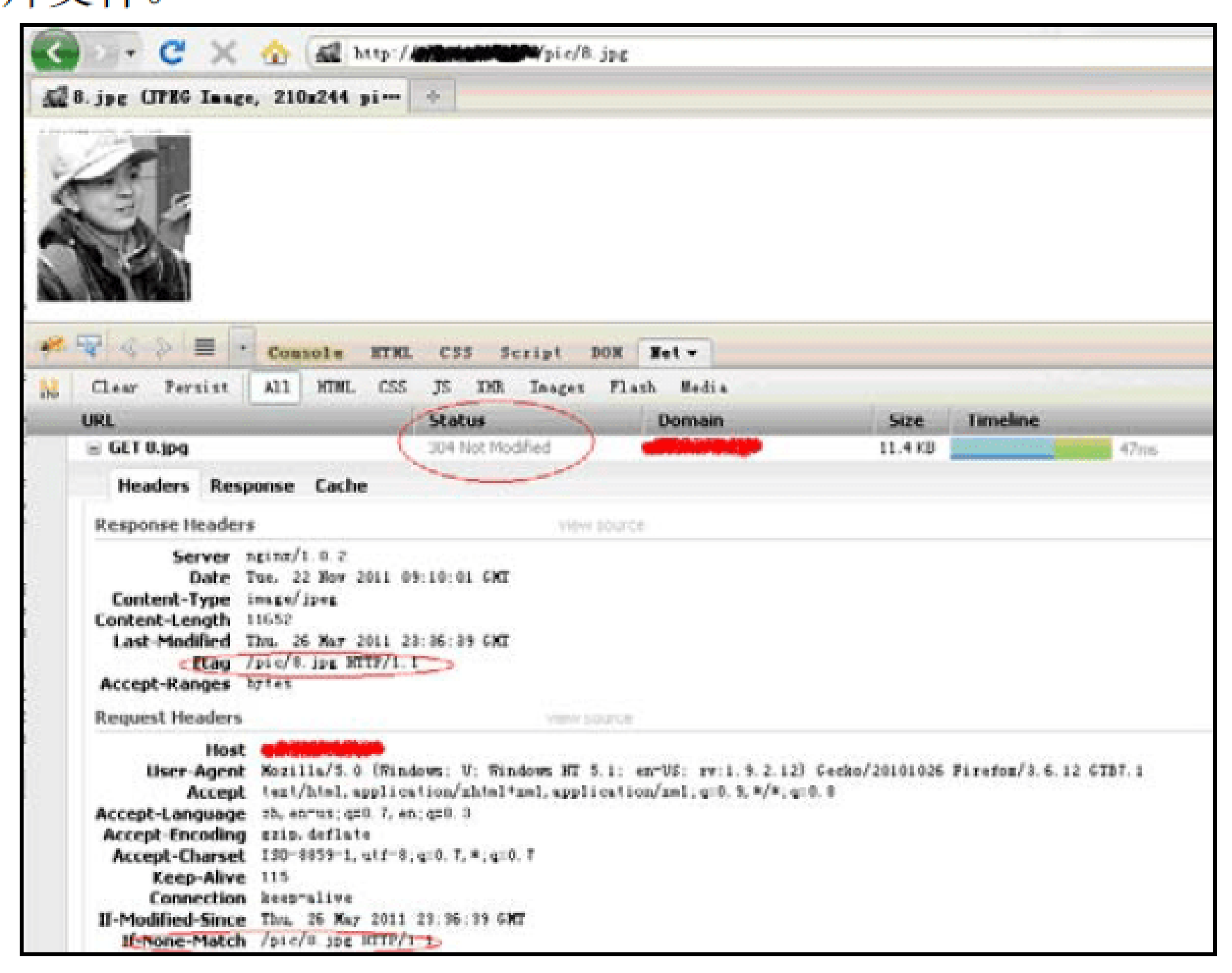


例 3：访问 css 文件。

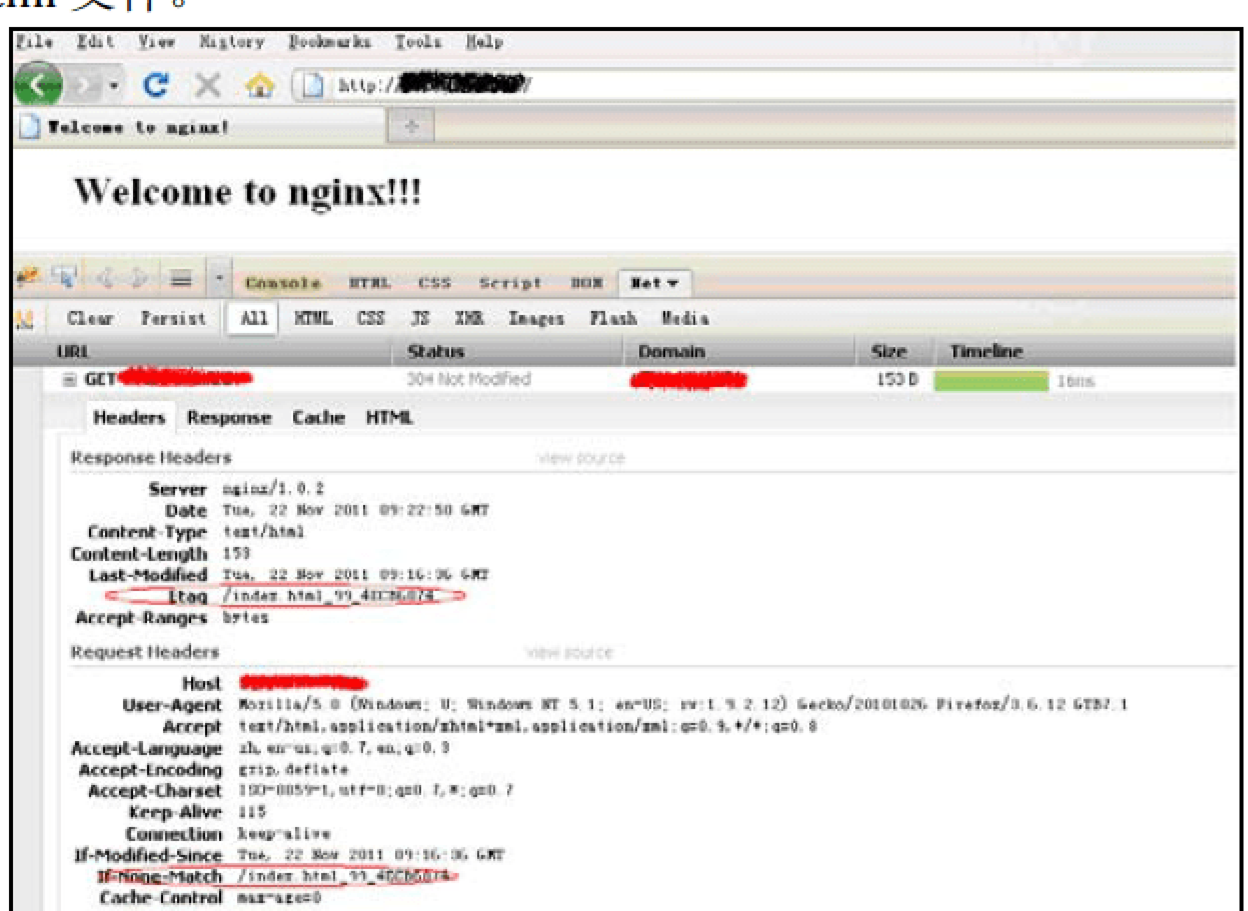


再次访问

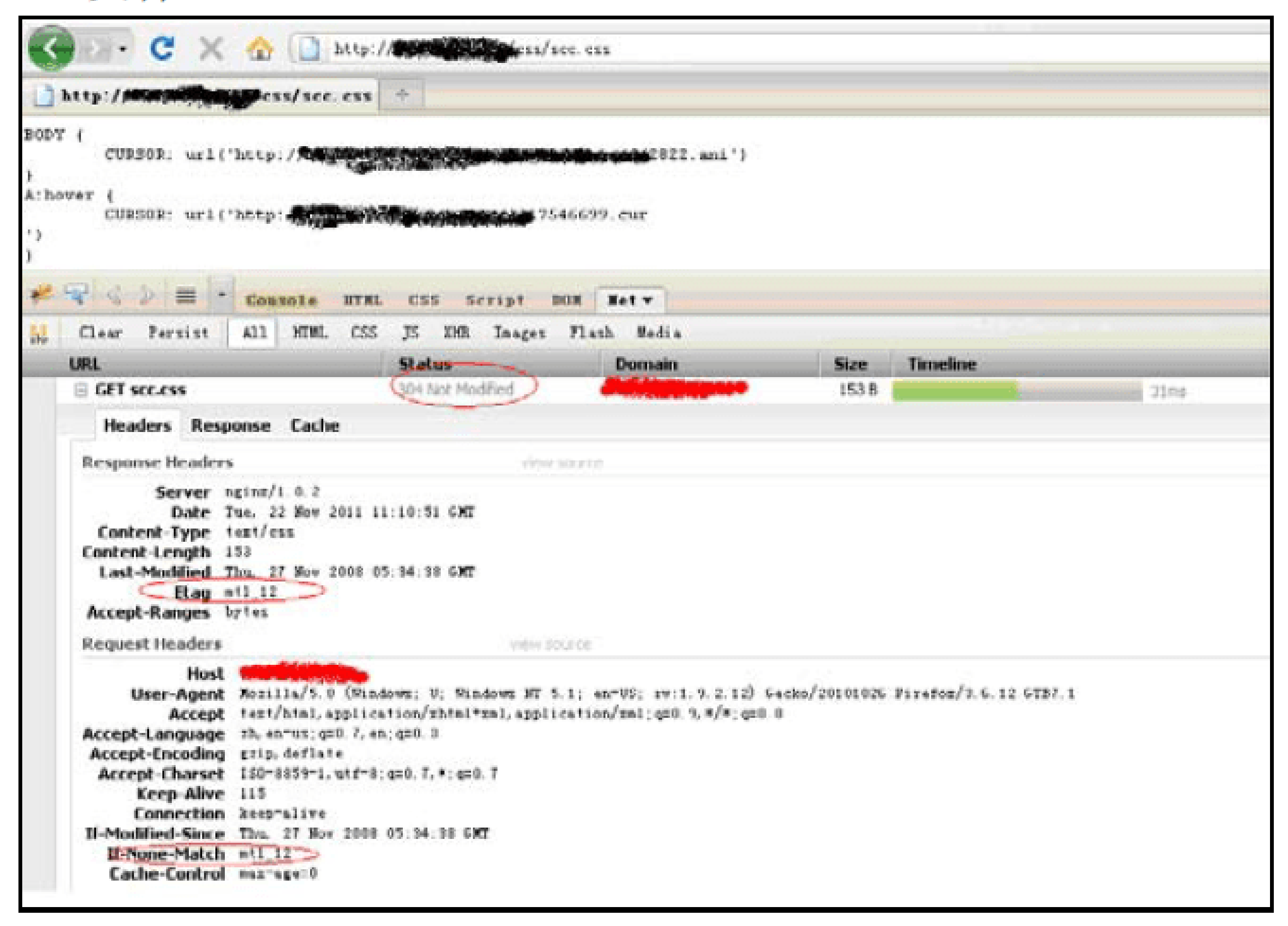
例 4：访问图片文件。



例 5：访问 html 文件。



例 6： 访问 css 文件。



45.2 安装 nginx-dynamic-etags 模块

下载并解压 nginx-dynamic-etags 模块。共有三个文件：

```
[root@mail nginx-dynamic-etags]# tree
.
|-- README
|-- config
'-- ngx http dynamic etags module.c

0 directories, 3 files
```

编译安装

```
[root@mail  nginx-1.0.2]#  ./configure  --prefix=/usr/local/nginx-1.0.2-
dynamic etags  \
> --add-module=/root/nginx-dynamic-etags
```

1. 配置示例

```
location / {
...
dynamic etags on;
...
}
```

2. 指令

从 nginx-dynamic-etags 模块的源代码来看，该模块只提供了一条指令。

指令名称：dynamic_etags

默认值：off

功能：该指令用于对动态网页添加 Etag 头。指令的取值是一个开关值，即 on 和 off。如果设置为 on，表示开启该功能；如果设置为 off，表示关闭该功能。

以下是这条指令在源代码中的定义：

```
static ngx_command_t  ngx_http_dynamic_etags_commands[] = {
{ ngx_string( "dynamic_etags" ),
  NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_FLAG,
  ngx_conf_set_flag_slot,
  NGX_HTTP_LOC_CONF_OFFSET,
  offsetof( ngx_http_dynamic_etags_loc_conf_t, enable ),
  NULL },
  ngx_null_command
};
```

3. 使用实例

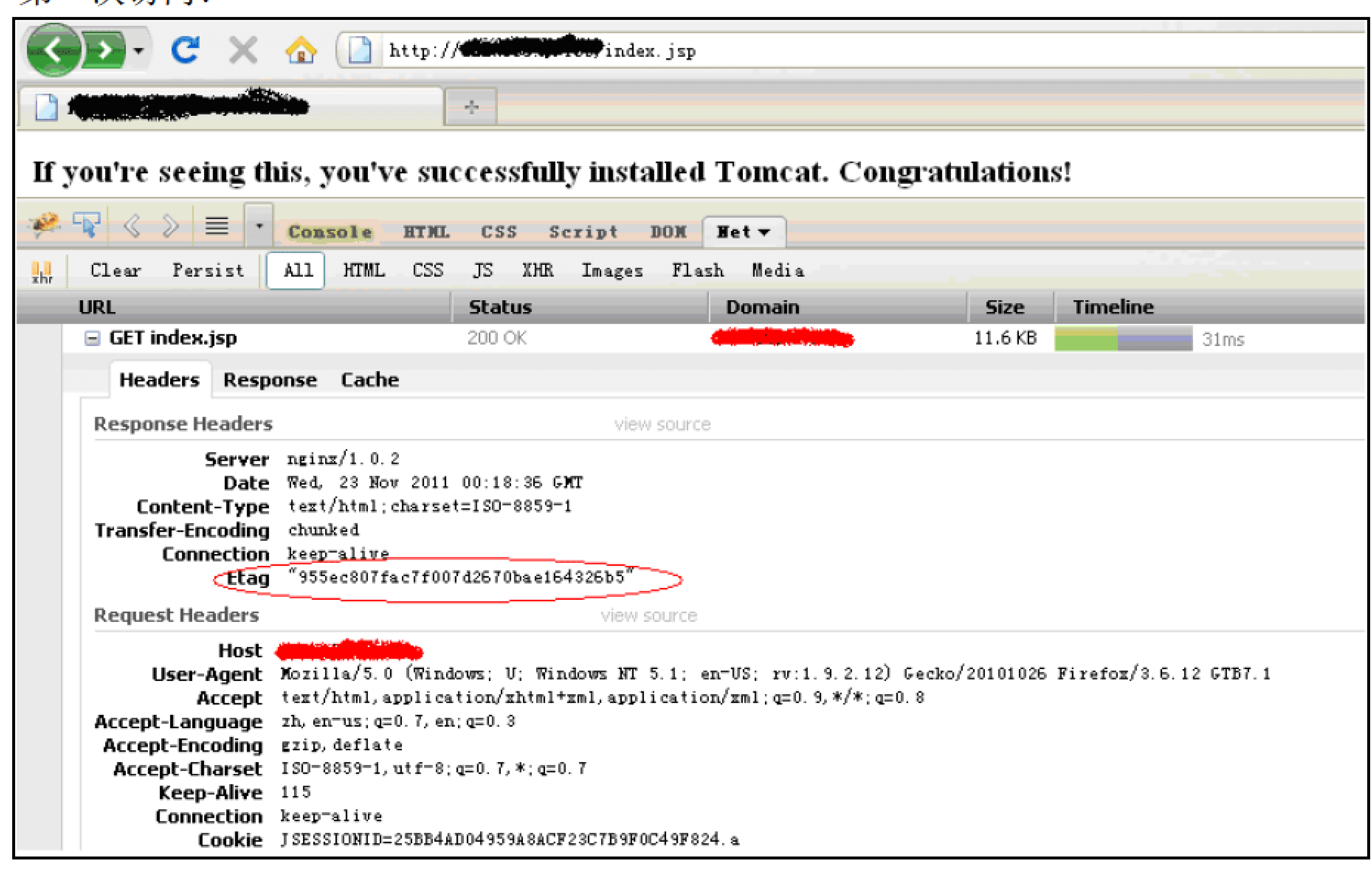
```
location ~* \.jsp$ {
 dynamic_etags on;
 proxy_passhttp://192.168.3.139:8080;
  }
```

这是一个简单的配置，是 Tomcat 与 Nginx 的结合，有关这部分内容在卷 2 中有详细的讲述。

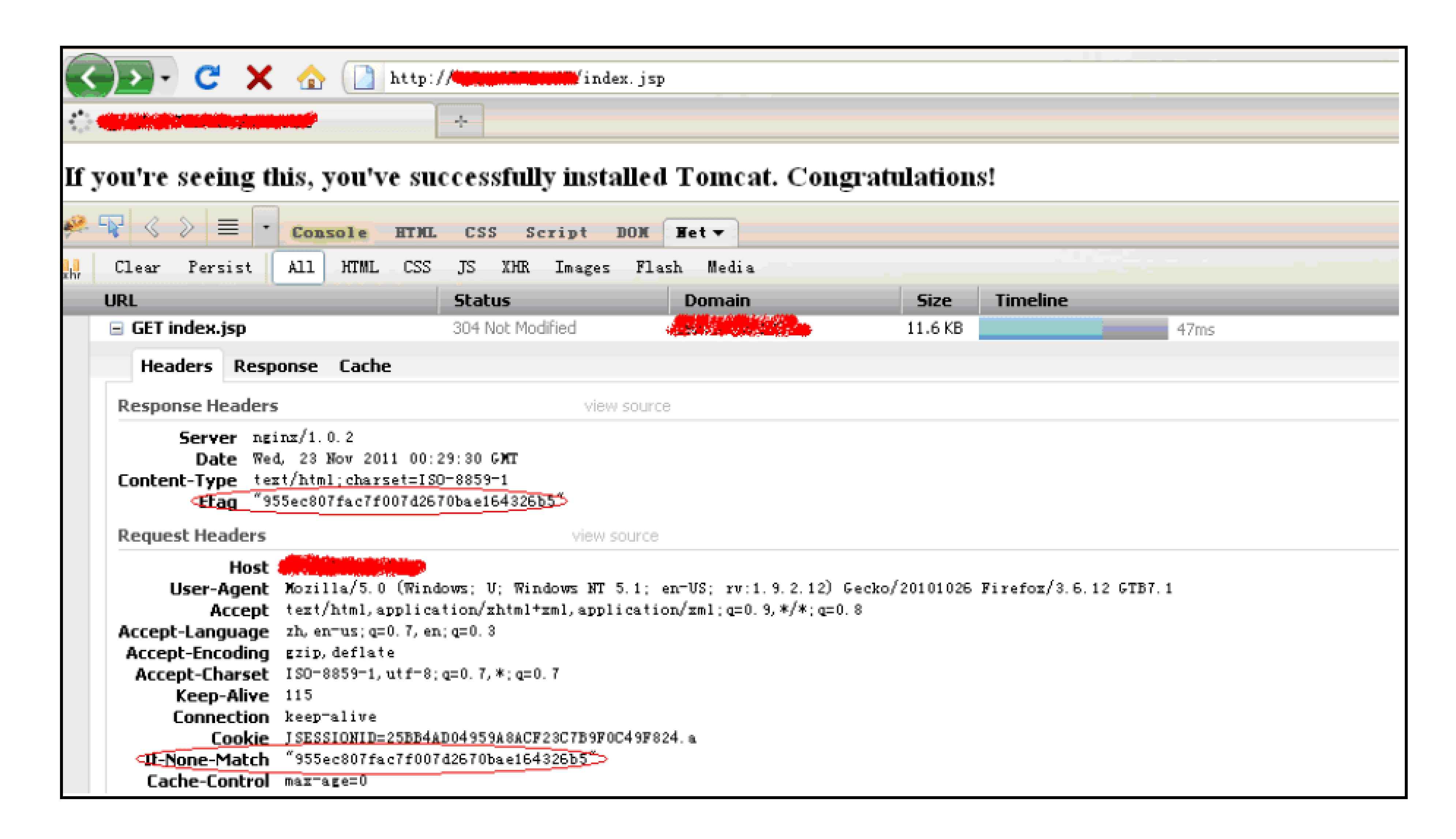
4. 访问测试

下面的测试分为两种情况：一种情况是在没有使用 Etag 之前，另一种情况是在使用 Etag 之后。

第一次访问：



再次访问：



45.3 四个头的区别与联系

Expires、Last-Modified、Cache-Control 和 Etag 是 HTTP/1.1 协议（由 RFC 2616 定义）中与网页缓存相关的 4 个头。它们的区别如下。

- Last-Modified 和 Etag：用来控制缓存的失效日期。
- Expires 和 Cache-Control：用来验证网页的有效性。

第 46 章　使用 upstream_keepalive 模块实现 keep-live

在使用代理模块、FastCGI 模块和 memcached 模块时，我们发现后台服务器和客户端传递完成数据后，Nginx 就会和后台服务器断开，每次都要重新连接，这样对于一个持续访问多个页面的客户端来说，无疑延长了等待时间，而对于 Nginx 服务器来说也多了每次请求连接和释放的过程，因此在这种情况下可以使用 upstream_keepalive 模块实现 keep-live。

1. 编译安装

下载 ngx_http_upstream_keepalive 模块：

```
    [root@mail ~]# wget http://mdounin.ru/hg/\
    > ngx http upstream keepalive/archive/tip.tar.gz
    --11:20:52--
http://mdounin.ru/hg/ngx http upstream keepalive/archive/tip.tar.gz
    Resolving mdounin.ru... 81.19.69.81
    Connecting to mdounin.ru|81.19.69.81|:80... connected.
    HTTP request sent, awaiting response... 200 Script output follows
    Length: unspecified [application/x-gzip]
    Saving to: 'ngx_http_upstream_keepalive-d9ac9ad67f45.tar.gz'

    [<=>] 10,371  16.4K/s   in 0.6s

    11:20:57 (16.4 KB/s) - 'ngx_http_upstream_keepalive-d9ac9ad67f45.tar.gz'
saved [10371]
```

解压 ngx_http_upstream_keepalive 模块：

```
    [root@mail ~]# tar -zxvf ngx http upstream keepalive-d9ac9ad67f45.tar.gz
    [root@mail ~]# cd ngx http upstream keepalive-d9ac9ad67f45
    [root@mail ngx http upstream keepalive-d9ac9ad67f45]# tree
    .
    |-- CHANGES
    |-- LICENSE
    |-- README
    |-- config
    |-- ngx http upstream keepalive module.c
    '-- t
    |-- fastcgi-keepalive.t
    |-- memcached-keepalive.t
    |-- proxy.t
    '-- stale.t
```

之所以要清点这个安装包是为了了解一下该模块的使用方法，读 README 就行了。后面我

们来分析一下它的使用方法。

另外，我们注意到还有个 t/目录，它下面放置的是 4 个测试程序，我们通过这 4 个测试程序也认识到，该模块能够应用到 memcached 模块、FastCGI 和代理模块。

编译安装：

```
[root@mail nginx-1.0.2]# ./configure --prefix=/usr/local/\
> nginx-1.0.2-upstream_keepalive
> --add-module=/root/ngx_http_upstream_keepalive-d9ac9ad67f45
[root@mail nginx-1.0.2]# make
[root@mail nginx-1.0.2]# make install
```

2. 配置示例

简单配置：

```
upstream memd {
server 127.0.0.1:11211;
server 10.0.0.2:11211;
keepalive 10;
}
```

在这个简单的配置中实际上使用的是轮询方式。

使用 IP 哈希：

```
upstream memd {
server 127.0.0.1:11211;
server 10.0.0.2:11211;
ip_hash;
keepalive 10;
}
```

用于 FastCGI：

```
http {

upstream backend {
server 127.0.0.1:8081;
keepalive 1024;
}

server {
listen   127.0.0.1:8080;
server_name  localhost;

location / {
fastcgi_pass backend;
fastcgi_keep_conn on;
}
}
```

```
}
```

用于代理（proxy）：

```
server {
listen   127.0.0.1:8080;
server name  localhost;

proxy_read_timeout 2s;
proxy_http_version 1.1;
proxy_set_header Connection "";

location / {
proxy pass http://backend;
}

location /unbuffered/ {
proxy_pass http://backend;
proxy_buffering off;
}

location /inmemory/ {
ssi on;
rewrite ^ /ssi.html break;
}
```

3. 指令

从以下源代码来看：

```
static ngx command t  ngx http upstream keepalive commands[] = {

{ ngx_string("keepalive"),
  NGX_HTTP_UPS_CONF|NGX_CONF_TAKE12,
  ngx http upstream keepalive,
  0,
  0,
  NULL },

  ngx_null_command
};
```

upstream_keepalive 模块只提供了一个指令，那就是 keepalive 指令，下面我们看一下这个指令。

指令名称：keepalive

语法：keepalive num [single]

使用环境：upstream

功能：该指令用于为 upstream 启用 keep-alive 功能。该指令有以下两个选项。

- num：该选项用于指定对 Memcached 的最大连接数，如果超过这个连接数，那么 Nginx 将会根据最近最少原则关闭掉连接进程。
- single：该选项表示将所有请求作为单个主机连接，使用这个连接标志将平等地对待不同的后台服务器。

该模块与标准的轮询能够一同进行工作，但是相信它也能够和其他复杂的负载均衡方式一同工作。但是需要注意的一点是，要在该模块使用之前使用相应的负载均衡方式。

例如：

```
upstream memcached {
  server 10.0.0.1:11211;
  server 10.0.0.2:11211;
  ip hash;
  keepalive 512;
}
```

在这个例子中，我们将“ip_hash”指令放置在了“keepalive 512”指令之前。

4. 配置实例

在 Nginx 服务器中添加以下配置：

```
upstream memcached {
  server 192.168.3.152:11211;
  server 192.168.3.153:11211;
  server 192.168.3.154:11211;
  keepalive 1024;
}

...

  location / {
set $memcached_key $uri;
memcached_pass memcached;
memcached_buffer_size 16k;
memcached_read_timeout 30000;
memcached send timeout 30000;
default type   text/html;
error page 404 @fallback;
  }

  location @fallback {
proxy_pass http://192.168.3.170:8080;
  }
```

访问测试

- 未使用 keeplive 指令之前的连接情况：

```
[root@mail nginx-dynamic-etags]# lsof -i:11211
COMMAND PID  USER   FD   TYPE  DEVICE SIZE NODE NAME
memcached 18944 nobody  36u  Ipv6 8489091  TCP *:11211 (LISTEN)
memcached 18944 nobody  37u  Ipv4 8489092  TCP *:11211 (LISTEN)
memcached 18944 nobody  38u  Ipv6 8489096  UDP *:11211
memcached 18944 nobody  39u  Ipv4 8489097  UDP *:11211
```

可见，在每次访问完成之后就断开了连接。

- 使用了 keeplive 之后的连接情况：

```
[root@mail nginx-dynamic-etags]# lsof -i:11211
COMMAND PID  USER   FD   TYPE  DEVICE SIZE NODE NAME
memcached 18944 nobody  36u  Ipv6 8489091  TCP *:11211 (LISTEN)
memcached 18944 nobody  37u  Ipv4 8489092  TCP *:11211 (LISTEN)
memcached 18944 nobody  38u  Ipv6 8489096  UDP *:11211
memcached 18944 nobody  39u  Ipv4 8489097  UDP *:11211
memcached 18944 nobody  40u  Ipv4 8489739  TCP mail.tt.com:11211
->mail.tt.com:43203 (ESTABLISHED)
nginx 19128 nobody  10u  Ipv4 8489738  TCP mail.tt.com:43203
->mail.tt.com:11211 (ESTABLISHED)
```

可见在请求的数据传输完成之后仍然保持着连接。

第 47 章　后台服务器的健康检测

healthcheck 模块是 Nginx 服务器的一个第三方模块插件，它通过访问后台指定的一个文件，如果返回的 HTTP 响应代码为 200+，则表示后端服务器是“好”的，否则则标记它们为“坏”的，它的工作原理类似于 haproxy、varnish 健康检测。

1. 编译安装

下载 healthcheck 模块：

```
[root@mail ~]# wget https://nodeload.github.com/   \
> cep21/healthcheck_nginx_upstreams/tarball/master
--14:36:55--
https://nodeload.github.com/cep21/healthcheck nginx upstreams/tarball/
   master
Resolving nodeload.github.com... 207.97.227.252
Connecting to nodeload.github.com|207.97.227.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13141 (13K) [application/octet-stream]
Saving to: 'cep21-healthcheck_nginx_upstreams-8870d34.tar.gz'

100%[===================================>] 13,141  28.9K/s   in 0.4s

14:36:59 (28.9 KB/s) - 'cep21-healthcheck_nginx_upstreams-8870d34.tar.gz'
saved [13141/13141]
```

解压后我们看一下包中的内容：

```
[root@mail cep21-healthcheck nginx upstreams-8870d34]# tree
.
|-- README
|-- config
|-- nginx.patch
|-- ngx_http_healthcheck_module.c
|-- ngx_http_healthcheck_module.h
'-- sample_ngx_config.conf

0 directories, 6 files
```

通过这些文件来了解该模块的使用。在包中有一个.pach 文件，因此它的安装方式和前面的第三方模块不一样，首先需要打补丁，安装步骤如下：

```
[root@mail nginx-1.0.8]# patch -p1 < /root/cep21-healthcheck_nginx
_upstreams-8870d34/nginx.patch
patching file src/http/ngx_http_upstream.c
Hunk #1 succeeded at 4270 (offset 3 lines).
```

```
patching file src/http/ngx_http_upstream.h
patching file src/http/ngx_http_upstream_round_robin.c
Hunk #2 succeeded at 14 with fuzz 2.
Hunk #3 succeeded at 35 (offset 9 lines).
Hunk #5 succeeded at 389 (offset 12 lines).
Hunk #7 succeeded at 496 (offset 12 lines).
Hunk #9 succeeded at 632 (offset 12 lines).
Hunk #10 succeeded at 645 with fuzz 1 (offset 4 lines).
patching file src/http/ngx_http_upstream_round_robin.h
```

编译安装：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8
-healthcheck\
> --add-module=/root/cep21-healthcheck nginx upstreams-8870d34
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

2. 配置示例

```
worker processes 5;
#daemon off;

events {
  worker_connections 1000;
}

# Only if you want to see lots of spam
error log log/error log debug http;

http {

  upstream test_upstreams {
server localhost:11114;
server localhost:11115;
hash $filename;
hash again 10;
healthcheck_enabled;
healthcheck_delay 1000;
healthcheck_timeout 1000;
healthcheck_failcount 1;
# Important: There is no \n at the end of this.  Or \r.  Make sure you
# don't have a \n or \r or anything else at the end of your healthcheck
# response
healthcheck_expected 'I_AM_ALIVE';
# Important: HTTP/1.0
```

```
  healthcheck_send "GET /health HTTP/1.0" 'Host: www.mysite.com';
  # Optional supervisord module support
  #supervisord none;
  #supervisord inherit backend status;
    }

    server {
  listen 11114;
  location / {
    root html_11114;
  }
    }
    server {
  listen 11115;
  location / {
    root html_11115;
  }
    }

    server {
  listen 81;

  location / {
    set $filename $request_uri;
    if  ($request_uri ~* ".*/ (.*) ")  {
  set $filename $1;
    }
    proxy set header Host $http host;
    proxy pass http://test upstreams;
    proxy connect timeout 3;
  }
  location /stat {
    healthcheck_status;
  }
}
  }
```

该示例配置就是模块中提供的 sample_ngx_config.conf 文件内容。

3. 指令

从下面的代码中可以看到 healthcheck 模块提供的指令：

```
static ngx command t  ngx http healthcheck commands[] = {
/**
 * If mentioned, enable healthchecks for this upstream
```

```
 */
{ ngx_string("healthcheck_enabled") ,
NGX_HTTP_UPS_CONF|NGX_CONF_NOARGS,
ngx http healthcheck enabled,
0,
0,
NULL },
/**
 * Delay in msec between healthchecks for a single peer
 */
{ ngx_string("healthcheck_delay") ,
NGX HTTP UPS CONF|NGX CONF TAKE1,
ngx http healthcheck delay,
0,
0,
NULL } ,
/**
 * How long in msec a healthcheck is allowed to take place
 */
{ ngx_string("healthcheck_timeout") ,
NGX HTTP UPS CONF|NGX CONF TAKE1,
ngx http healthcheck timeout,
0,
0,
NULL },
/**
 * Number of healthchecks good or bad in a row it takes to switch from
 * down to up and back.  Good to prevent flapping
 */
{ ngx_string("healthcheck_failcount") ,
NGX_HTTP_UPS_CONF|NGX_CONF_TAKE1,
ngx_http_healthcheck_failcount,
0,
0,
NULL } ,
/**
 * What to send for the healthcheck.  Each argument is appended by \r\n
 * and the entire thing is suffixed with another \r\n.  For example,
 *
 * healthcheck_send 'GET /health HTTP/1.1'
 *   'Host: www.facebook.com' 'Connection: close';
 *
 * Note that you probably want to end your health check with some directive
```

```
 * that closes the connection, like Connection: close.
 *
 */
{ ngx string ("healthcheck send") ,
NGX HTTP UPS CONF|NGX CONF 1MORE,
ngx http healthcheck send,
0,
0,
NULL },
/**
 * What to expect in the HTTP BODY, (meaning not the headers) , in a correct
 * response
 */
{ ngx_string ("healthcheck_expected") ,
NGX HTTP UPS CONF|NGX CONF TAKE1,
ngx_http_healthcheck_expected,
0,
0,
NULL },
/**
 * How big a buffer to use for the health check.  Remember to include
 * headers PLUS body, not just body.
 */
{ ngx_string ("healthcheck_buffer") ,
NGX_HTTP_UPS_CONF|NGX_CONF_TAKE1,
ngx_http_healthcheck_buffer,
0,
0,
NULL },
/**
 * When inside a /location block, replaced the HTTP body with backend
 * health status.  Use similarly to the stub_status module
 */
{ ngx string ("healthcheck status") ,
  NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF NOARGS,
  ngx http set healthcheck status,
  0,
  0,
  NULL },
ngx_null_command
};
```

指令名称：healthcheck_enabled

语法：healthcheck_enabled

使用环境：upstream

功能：在 upstream 区段中使用该指令表示使用该模块。

指令名称：healthcheck_delay

语法：healthcheck_delay num

默认值：10000

使用环境：upstream

功能：该指令用于设置对后台检查的时间间隔，单位为毫秒（msec）。

指令名称：healthcheck_timeout

语法：healthcheck_timeout num

默认值：2000

使用环境：upstream

功能：该指令用于设置检测时长，即在这个时间内没有响应则被认为是超时。

指令名称：healthcheck_failcount

语法：healthcheck_failcount num

默认值：2

使用环境：upstream

功能：该指令用于设置标志为一个好的或者是坏的后台节点的健康检测次数，如果达到这个值，那么将该节点设置为“down”或者是“up”。

指令名称：healthcheck_send

语法：healthcheck_send

默认值：null

使用环境：upstream

功能：该指令指定所需的指令，就是说发送什么样的指令来检测健康状况。

例如：

```
healthcheck send 'GET /health HTTP/1.0'
 'Host: www.yourhost.com';
```

指令名称：healthcheck_expected

语法：healthcheck_expected

默认值：<UNSET>

使用环境：upstream

功能：该指令用于在一个正确的响应中指定在 HTTP BODY 中期望的内容，如果没有设置，那么对于一个点只需要 HTTP 200 状态代码。

指令名称：healthcheck_buffer

语法：healthcheck_buffer

默认值：1000

使用环境：upstream

功能：该指令用于设置健康检测所使用的缓存大小。注意，包括头（headers）和体（body），而不仅仅是 body。

指令名称：healthcheck_status

语法：healthcheck_status

使用环境：upstream

功能：它的功能类似于 stub_status 模块。如果在一个/location 区段使用该指令时，那么它将显示后台的健康状况。

4. 配置实例

在 Nginx 的配置文件中添加以下配置内容：

```
  upstream tomcat  {
server 192.168.3.141:8080;
server 192.168.3.142:8080;
healthcheck_enabled;
healthcheck_delay 1000;
healthcheck_timeout 1000;
healthcheck failcount 1;
healthcheck send "GET /test.txt HTTP/1.0" ;
  }

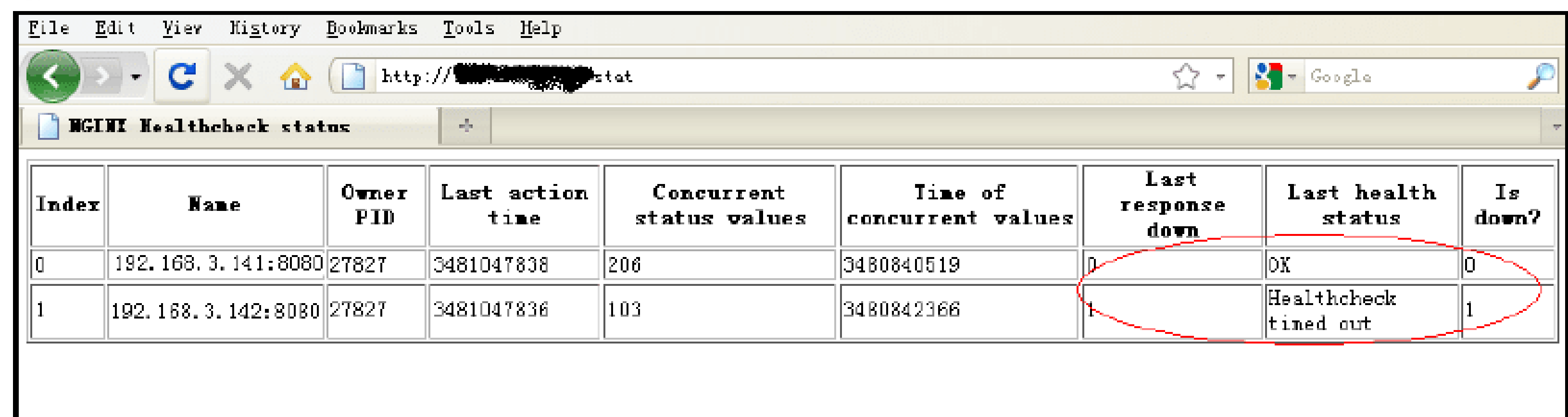
server {
listen   80;
server_name  www.xx.cn;

  location / {
proxy pass http://tomcat;
  }

  location /stat {
healthcheck_status;
  }
}
```

5. 访问测试

访问 http://www.xx.cn/stat:



Index	Name	Owner PID	Last action time	Concurrent status values	Time of concurrent values	Last response down	Last health status	Is down?
0	192.168.3.141:8080	27827	3481047838	206	3480840519	0	OK	0
1	192.168.3.142:8080	27827	3481047836	103	3480842366	1	Healthcheck timed out	1

清楚地报告了当前的状态。

我们可以根据具体的情况来设置检测时间，然后再进行“down”和“up”后台服务器的测试，在此就不多讲了，关键在于实践中的控制。

第 48 章　使用 sticky 模块实现粘贴性会话

使用由“IP 级别”（比如由 ip_hash 实现的“粘贴性”）实现的“粘贴性”（就是请求总是被转发到同一台后台服务器）在有些环境中并不是一个好的方法。例如，当一个客户端所在的环境是在代理下或者是 NAT 上网，等等。在这些情况下如果使用 ip_hash，那么将不会实现真正意义上的负载均衡，因此有必要使用“会话级”及 cookie 级的负载均衡。使用 cookie 时，它将会为每个浏览器实现唯一的标识，当 sticky 模块不能被应用时，它将切换回 RR（Round Robin）模式，如果客户端浏览器不支持 cookie，那么该模块将不能使用。

特别要说明的一点是，这个模块不能和 ip_hash 在同一个 upstream 区段使用。

1. 编译安装

下面我们来安装 sticky 模块。

下载 sticky 模块：

```
[root@mail ~]# wget http://nginx-sticky-module.googlecode.com\
> /files/nginx-sticky-module-1.0.tar.gz
--18:03:17--
http://nginx-sticky-module.googlecode.com/files/nginx-sticky-module-1.0.ta
r.gz
Resolving nginx-sticky-module.googlecode.com... 74.125.71.82
Connectingtonginx-sticky-module.googlecode.com|74.125.71.82|:80...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 114184 (112K) [application/octet-stream]
Saving to: 'nginx-sticky-module-1.0.tar.gz'

100%[========================>] 114,184  150K/s   in 0.7s

18:03:19(150 KB/s)-'nginx-sticky-module-1.0.tar.gz' saved [114184/114184]
```

解压安装包：

```
[root@mail ~]# tar -zxvf nginx-sticky-module-1.0.tar.gz
 [root@mail nginx-sticky-module-1.0]# tree
.
|-- LICENSE
|-- README
|-- config
|-- docs
|   |-- sticky.pdf
|   '-- sticky.vsd
```

```
|-- ngx_http_sticky_misc.c
|-- ngx_http_sticky_misc.h
'-- ngx_http_sticky_module.c

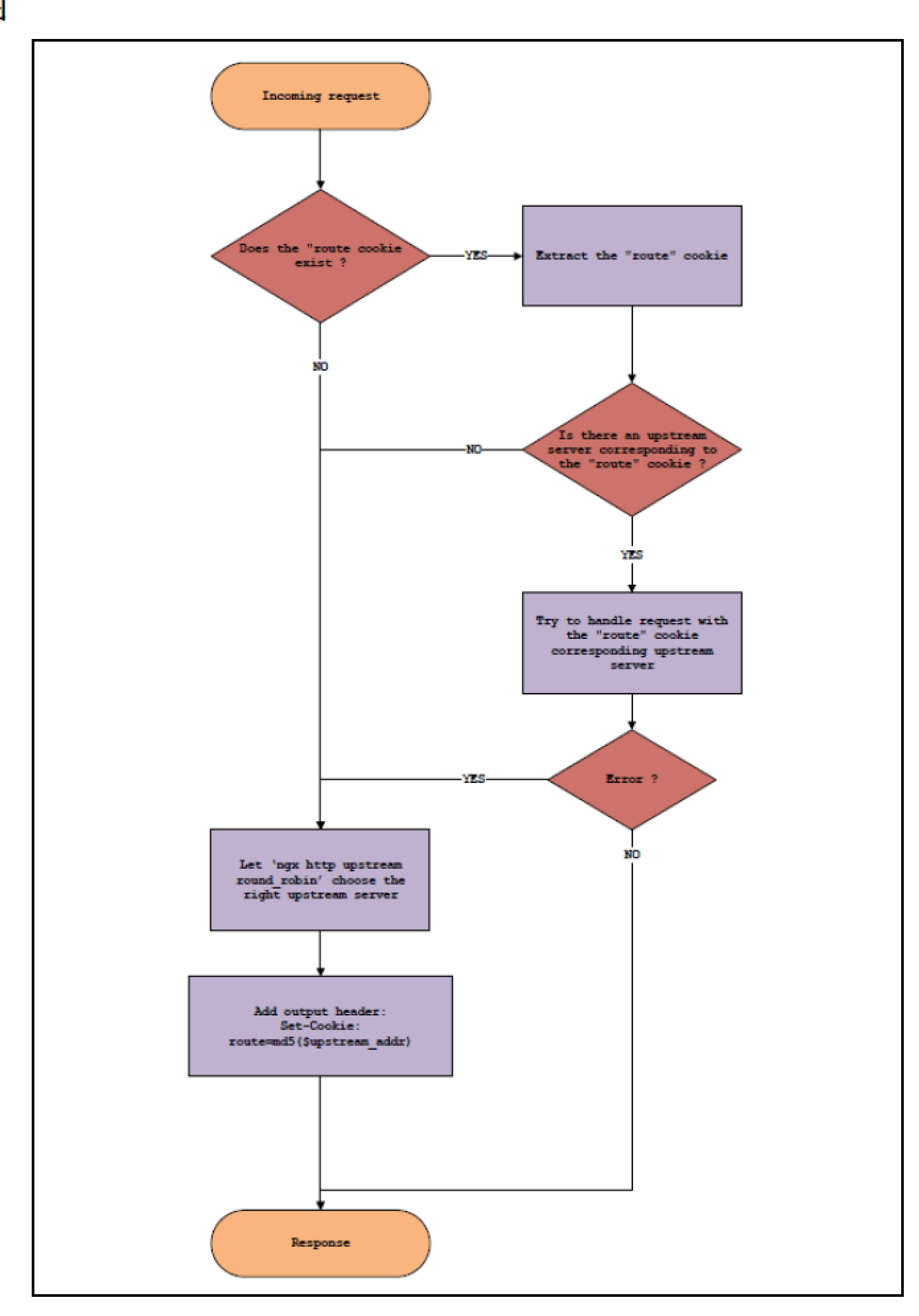
1 directory, 8 files
```

我们在解压后台的目录中看到，除了 sticky 代码之外，还有 README 和 sticky.pdf 文件，有必要首先看一下这两个文件。

编译安装 sticky：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8
-sticky-module --add-module=/root/nginx-sticky-module-1.0/
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

2. 示意图



3. 序列图

```
 (client)  (nginx) (upstream servers)
>-- GET /URI1 HTTP/1.0 -----------> |
| *** nginx choose one upstream by RR ***
| >----- GET /URI1 HTTP/1.0   ----> |
| <------- HTTP/1.0 200 OK -------< |
<-- HTTP/1.0 200 OK --------------< |
Set-Cookie: route=md5(upstream)  |
|
>-- GET /URI2 HTTP/1.0 -----------> |
Cookies: route  |
| *** nginx redirect to "route" ***
| >----- internal fetch /URI2 ----> |
| <--- internal response /URI2 ---< |
<-- HTTP/1.0 200 OK --------------< |
  (…)
```

4. 配置示例

```
upstream {
sticky;
server 127.0.0.1:9000;
server 127.0.0.1:9001;
server 127.0.0.1:9002;
}
```

5. 指令

从源代码来看：

```
static ngx command t  ngx http sticky commands[] = {

{ ngx_string("sticky"),
NGX_HTTP_UPS_CONF|NGX_CONF_ANY,
ngx_http_sticky_set,
0,
0,
NULL },

ngx_null_command
};
```

只有一条指令，但是它的选项却不少。

指令名称：sticky

语法：sticky [name=route] [domain=.foo.bar] [path=/][expires=1h] [hash=index|md5|sha1]

[no_fallback];

使用环境：upstream

功能：该模块的功能在于能够实现在 Nginx 服务器中将请求转发到后台服务器时实现粘贴性会话，也就是添加一个 cookie，总是将请求转发到同一台后台服务器。使用 cookie 来跟踪后端服务器，它使得每个浏览器都有唯一标识。

sticky 指令相关选项的使用如下。

- name：该参数用于指定 cookie 的名字，即用于持久跟踪上游服务器的 cookie 名字。
  默认值：route
- domain：该参数用于设置使用 cookie 的有效域。
  默认值：无
- path：该参数用于设置使用 cookie 有效的路径。
  默认值：无
- expires：该参数用于设置 cookie 的有效使用时间，但设置的时间必须大于 1 秒。
  默认值：无
- hash：该参数用于设置上游服务器（upstream server）编码，不能够使用 hmac。
  - md5|sha1：众所周知的hash类型，在此不再讲述；
  - index：它不是一个hash值，而是使用内存中的index，这种方式速度较快并且开销也小。

  默认值：md5
- hmac：哈希机 hmac 制用于对上游服务器（upstream server）编码，它类似于 hssh 机制，但是它使用的是 hmac_key，它是安全的散列值。不能够用于 hash。
  默认值：无
- hmac_key：该参数用于指定 hmac 使用的 key，如果设置了 hmac，那么必须设置该参数。
  默认值：无
- no_fallback：如果设置了该参数，那么在请求一同提供的 cookie 的相应后台服务器无效时，将会返回 502（就是说网关或者是代理错误）。

6. 配置实例

在 Nginx 的配置文件中添加以下配置：

```
upstream apache{
sticky;
    server 192.168.3.121:80;
    server 192.168.0.122:80;
}
```

7. 访问测试

访问测试比较简单，就不再举例了。另外还可以参考卷 2 中“Nginx 与 Java”部分中 nginx-upstream-jvm-route 模块的功能和用法。

第 49 章　Nginx 对后台服务器实现“公平”访问

该模块的功能在于其将进入的请求转发到一个最近最少“忙”的后台服务器，而不是使用 RR（轮询）方式。它是一个用于对后端代理服务器实现公平“工作”的模块，它增强了标准的 RR 方式负载均衡，通过跟踪“忙”的后台服务器（例如 Thin，Ebb，Mongrel）来均衡地载入不“忙”的后台服务器进程。

下面我们来下载并且安装 upstream-fair 模块。

下载 upstream-fair 模块：

```
[root@mail ~]# wget https://nodeload.github.com/gnosek/nginx-upstream
-fair/tarball/master
--08:37:53--
https://nodeload.github.com/gnosek/nginx-upstream-fair/tarball/master
Resolving nodeload.github.com... 207.97.227.252
Connecting to nodeload.github.com|207.97.227.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10066 (9.8K) [application/octet-stream]
Saving to: 'gnosek-nginx-upstream-fair-7171df8.tar.gz'

100%[=================================>] 10,066  14.4K/s   in 0.7s

08:37:56 (14.4 KB/s) - 'gnosek-nginx-upstream-fair-7171df8.tar.gz' saved
[10066/10066]
```

查看目录结构：

```
[root@mail ~]# tar -zxvf gnosek-nginx-upstream-fair-7171df8.tar.gz
[root@mail gnosek-nginx-upstream-fair-7171df8]# tree
.
|-- README
|-- config
'-- ngx http upstream fair module.c

0 directories, 3 files
```

目录结构很简单，需要查看一下 README 文件。

编译安装：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local/nginx-1.0.8-fair
--add-module=/root/gnosek-nginx-upstream-fair-7171df8
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

1. 配置示例

```
upstream mongrel {
fair;
server 127.0.0.1:5000;
server 127.0.0.1:5001;
server 127.0.0.1:5002;
  }
```

2. 指令

查看源代码：

```
static ngx command t  ngx http upstream fair commands[] = {

{ ngx_string ("fair") ,
  NGX_HTTP_UPS_CONF|NGX_CONF_ANY,
  ngx http upstream fair,
  0,
  0,
  NULL },

{ ngx_string ("upstream_fair_shm_size") ,
  NGX_HTTP_MAIN_CONF|NGX_CONF_TAKE1,
  ngx_http_upstream_fair_set_shm_size,
  0,
  0,
  NULL },

  ngx_null_command
};
```

可见 upstream-fair 模块提供了以下两条指令。

指令名称：fair

语法：fair

使用环境：upstream

功能：启用“公平”功能。

指令名称：upstream_fair_shm_size

语法：upstream_fair_shm_size size

默认值：default upstream_fair_shm_size 32k

使用环境：http

功能：该指令用于设置存储有关繁忙后台服务器信息的共享内存大小。默认值是 8 个内存页面，因此在大多数系统上是 32KB。

3. 配置实例

```
http {

…

upstream_fair_shm_size 64k;

}

upstream mongrel {
fair;
server 127.0.0.1:5000;
server 127.0.0.1:5001;
server 127.0.0.1:5002;
  }
```

4. 访问测试

该模块的访问测试比较简单，在此就不再介绍了。

第 50 章　Nginx 使用 redis 数据库

在实际的生产环境中，redis 数据库已经使用很广，因为它的性能卓越，因此我们有必要掌握 Nginx 与 redis 数据库的结合。

本章我们来学习两个与 Nginx 服务器相关的 redis 模块。

这两个模块都是用于 upstream 区段的，如果需要 Nginx 与 redis 服务器进行“沟通”时，那么就需要这两个模块。其中第二个模块即我们所说的 redis2，使用的非阻塞方式更适用于 redis 2.x 版本，支持完整的 redis 2.0 标准协议的执行，包括对 redis 流水线操作（pipelining）的支持。

redis 模块能够从 redis 服务器返回原始的 TCP 响应。推荐使用 LuaRedisParser 模块（该模块由纯 C 编写）来解析这些响应，如果与 HttpLuaModule 组合将会把这些响应解释为 lua 数据格式。

而如果仅使用 get redis 指令，那么可以使用 HttpRedisModule 模块，它将返回 redis 响应中解析的部分内容，因为它只需要执行 get 命令。

50.1　安装 redis 模块

下载并安装 redis 模块：

```
[root@mail ~]# wget http://people.FreeBSD.org/~osa/ngx_http_redis
-0.3.5.tar.gz
[root@mail ~]# tar -zxvf ngx http redis-0.3.5.tar.gz
[root@mail ~]# cd ngx http redis-0.3.5
[root@mail ngx_http_redis-0.3.5]# tree
.
|-- AUTHOR
|-- CHANGES
|-- LICENSE
|-- README
|-- config
|-- ngx http redis module.c
'-- t
|-- Test
|   '-- Nginx.pm
|-- redis.t
'-- redis db not set.t

2 directories, 9 files
```

在这个目录结构中，README 文件是要重点看一下的。另外 ngx_http_redis_module.c 也要看一下，最起码指定该模块提供的可用命令。目录 t/ 下是测试程序。

```
 [root@mail nginx-1.0.8]# ./configure --prefix=/usr/local//usr/local
/nginx-1.0.8-redis/   \
> --add-module=/root/ ngx_http_redis-0.3.5
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

1. 配置示例

例 1

```
http
{
…
server {
location / {
set $redis key  "$uri?$args";
redis_pass  127.0.0.1:6379;
error_page  404 502 504 = @fallback;
}

location @fallback {
proxy pass  backed;
}
}
}
```

例 2

```
http
{
…
upstream redis {
server  127.0.0.1:6379;
}

server {
…
location / {

eval_escalate on;

eval $answer {
set $redis key  "$http user agent";
redis pass  redis;
}
```

```
    proxy_pass $answer;
    }
    …
     }
    }
```

2. 指令

查看该模块的源代码部分：

```
    static ngx_command_t  ngx_http_redis_commands[] = {

    { ngx_string("redis_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_redis_pass,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    #if defined nginx_version && nginx_version >= 8022
    { ngx_string("redis_bind"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_http_upstream_bind_set_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis_loc_conf_t, upstream.local),
      NULL },
    #endif

    { ngx_string("redis_connect_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("redis_send_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis_loc_conf_t, upstream.send_timeout),
      NULL },
```

```
    { ngx_string ("redis_buffer_size") ,

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx conf set size slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof (ngx_http_redis_loc_conf_t, upstream.buffer_size) ,
      NULL },

    { ngx_string ("redis_read_timeout") ,

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx conf set msec slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof (ngx_http_redis_loc_conf_t, upstream.read_timeout) ,
      NULL },

    { ngx_string ("redis_next_upstream") ,

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF 1MORE,
      ngx conf set bitmask slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof (ngx_http_redis_loc_conf_t, upstream.next_upstream) ,
      &ngx_http_redis_next_upstream_masks },

      ngx_null_command
    };
```

可以看出使用的指令如下。

指令名称：redis_pass

语法：redis_pass [name:port]

默认值：none

使用环境：http，server，location

功能：该指令用于设置连接的后台 redis 服务器，可以使用 upstream 指令设置的集群名称。其设置格式为 IP（主机名）：端口号。redis 键为“/uri?args”。

指令名称：redis_bind

语法：redis_bind [addr]

默认值：none

使用环境：http，server，localtion

功能：使用指定的 IP 地址作为连接 redis 的源 IP 地址。

指令名称：redis_connect_timeout

语法：redis_connect_timeout [time]

默认值：60000

使用环境：http，server，location

功能：该指令指定连接 redis 的超时设置，单位为毫秒。

指令名称：redis_read_timeout

语法：redis_read_timeout [time]

默认值：60000

使用环境：http，server，location

功能：该指令用于设置读取 redis 的超时时间，单位为毫秒。

指令名称：redis_send_timeout

语法：redis_send_timeout [time]

默认值：60000

使用环境：http，server，location

功能：该指令用于设置向 redis 发送数据的超时时间，单位为毫秒。

指令名称：redis_buffer_size

语法：redis_buffer_size [size]

默认值：由系统决定。

使用环境：http，server，location

功能：该指令用于设置接收和发送的缓存大小，单位为字节。默认值的大小为一个内存页面大小，可以通过“getconf PAGESIZE”指令来获取所在系统的内存页面大小，一般为 4KB 或者是 8KB。

指令名称：redis_next_upstream

语法：redis_next_upstream [error | timeout | invalid_response | not_found | off]

默认值：error timeout

使用环境：http，server，location

功能：该指令用于设置由哪些条件（或者叫指标）发现连接的 redis 后台失败，在发现失败之后，会将该请求转发到其他的 redis 服务器。该指令仅在后台，或者说是使用 redis_pass 指令连接的上游服务器在两台以上才可以使用。

3. 变量

看以下源代码：

```
static ngx_str_t  ngx_http_redis_key = ngx_string("redis_key");
static ngx_str_t  ngx_http_redis_db  = ngx_string("redis_db")
```

可以看出，提供了以下两个变量。

变量名称：$redis_key

功能：该变量表示 redis 键的值。

变量名称：$redis_db

功能：该变量表示 redis 数据库的数量（小于 0.3.4 版本需要定义），对于 0.3.4 以上的版本

不必使用，如果没有定义该变量，那么默认值为“0”。

4. 配置实例

```
{
…
upstream tm {
server  192.168.3.139:8080;
}


server {
location / {
set $redis key  "$uri?$args";
redis pass  192.168.3.192:6379;
error page  404 502 504 = @fallback;
}

location @fallback {
proxy_pass  tm;
}
}
```

5. 访问测试

关于访问测试这里就不再进行了。

6. 配置实例

```
{
…
    upstream tm {
        server  192.168.3.139:8080;
    }


        server {
                location / {
                        set $redis_key  $uri;
                        redis_pass     192.168.3.192:6379;
                        error_page     404 502 504 = @fallback;
                }

                location @fallback {
```

```
                    proxy_pass      tm;
                }
            }
```

7. 访问测试

通过 redis 的客户端添加缓存文件：

```
[root@mfs2 bin]# pwd
/usr/local/redis-2.4.3/bin
[root@mfs2 bin]# ./redis-cli
redis 127.0.0.1:6379> set /b.html abc
OK
redis 127.0.0.1:6379> get /b.html
"abc"
redis 127.0.0.1:6379>
```

访问 http://www.xx.cn/b.html



8. 支持 keep-live 功能

在 0.3.5 版本中支持了 keep-live 连接后台数据库的功能，源于 Nginx 1.1.4 版本的 ngx_http_memcached 模块。对于之前的版本，如果要使用 keep-live 功能，那么可以参考第三方模块 ngx_upstream_keepalive 来实现，这个模块我们在前面已经讲过。我们来看下面的一个配置示例：

```
  http {
upstream redisbackend {
  server 127.0.0.1:6379;

  # a pool with at most 1024 connections
  # and do not distinguish the servers:
  keepalive 1024 single;
}

server {
…
location /redis {
…
redis pass redisbackend;
}
}
```

```
    }
```

50.2 安装 redis2 模块

ngx_redis2 模块使用 redis 2.0 协议。

下载 ngx_redis2 模块：

```
[root@mail ~]# wget https://nodeload.github.com/agentzh/   \
> redis2-nginx-module/tarball/v0.08rc1
```

源代码目录结构：

```
[root@mail agentzh-redis2-nginx-module-23cf589]# tree
.
|-- Changes
|-- README
|-- README.markdown
|-- config
|-- doc
|   '-- HttpRedis2Module.wiki
|-- misc
|   '-- serv.erl
|-- src
|   |-- common.rl
|   |-- ddebug.h
|   |-- multi_bulk_reply.rl
|   |-- ngx_http_redis2_handler.c
|   |-- ngx_http_redis2_handler.h
|   |-- ngx_http_redis2_module.c
|   |-- ngx_http_redis2_module.h
|   |-- ngx_http_redis2_reply.c
|   |-- ngx_http_redis2_reply.h
|   |-- ngx_http_redis2_reply.rl
|   |-- ngx_http_redis2_util.c
|   '-- ngx_http_redis2_util.h
|-- t
|   |-- bugs.t
|   |-- eval.t
|   |-- pipeline.t
|   '-- sanity.t
|-- util
|   '-- build.sh
'-- valgrind.suppress

5 directories, 24 files
```

从这个目录结构我们可以看出，该源代码包含三部分内容，即帮助文件、模块代码和测试工具。

编译安装：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local\
> /nginx-1.0.8-redis2
> --add-module=/root/agentzh-redis2-nginx-module-23cf589/
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

1. 配置示例

```
location /foo {
set $value 'first';
redis2 query set one $value;
redis2 pass 127.0.0.1:6379;
}

# GET /get?key=some_key
location /get {
set_unescape_uri $key $arg_key;  # this requires ngx_set_misc
redis2 query get $key;
redis2 pass foo.com:6379;
}

# GET /set?key=one&val=first%20value
location /set {
set_unescape_uri $key $arg_key;  # this requires ngx_set_misc
set_unescape_uri $val $arg_val;  # this requires ngx_set_misc
redis2 query set $key $val;
redis2 pass foo.com:6379;
}

# multiple pipelined queries
location /foo {
set $value 'first';
redis2 query set one $value;
redis2 query get one;
redis2 query set one two;
redis2 query get one;
redis2_pass 127.0.0.1:6379;
}

location /bar {
# $ is not special here...
```

```
    redis2_literal_raw_query '*1\r\n$4\r\nping\r\n';
    redis2_pass 127.0.0.1:6379;
    }

    location /bar {
    # variables can be used below and $ is special
    redis2 raw query 'get one\r\n';
    redis2_pass 127.0.0.1:6379;
    }

    # GET /baz?get%20foo%0d%0a
    location /baz {
    set unescape uri $query $query string; # this requires the ngx set misc
module
    redis2 raw query $query;
    redis2_pass 127.0.0.1:6379;
    }
```

2. 指令

查看以下源代码：

```
    static ngx command t  ngx http redis2 commands[] = {

    { ngx_string ("redis2_query") ,
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_1MORE,
      ngx_http_redis2_query,
      NGX HTTP LOC CONF OFFSET,
      0,
      NULL },

    { ngx_string ("redis2_raw_query") ,
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_redis2_set_complex_value_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof (ngx_http_redis2_loc_conf_t, complex_query) ,
      NULL },

    { ngx_string ("redis2_raw_queries") ,
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE2,
      ngx_http_redis2_raw_queries,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },
```

```
    { ngx_string("redis2_literal_raw_query"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_str_slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, literal_query),
      NULL },


    { ngx_string("redis2_pass"),
      NGX_HTTP_LOC_CONF|NGX_HTTP_LIF_CONF|NGX_CONF_TAKE1,
      ngx_http_redis2_pass,
      NGX HTTP LOC CONF OFFSET,
      0,
      NULL },

    { ngx_string("redis2_bind"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx http upstream bind set slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.local),
      NULL },

    { ngx_string("redis2_connect_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx conf set msec slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("redis2_send_timeout"),

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx conf set msec slot,
      NGX HTTP LOC CONF OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("redis2_buffer_size"),

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx_conf_set_size_slot,
```

```
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.buffer_size),
      NULL },

    { ngx_string("redis2_read_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("redis2_next_upstream"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_redis2_loc_conf_t, upstream.next_upstream),
      &ngx_http_redis2_next_upstream_masks },

      ngx_null_command
    };
```

可以看出，该源代码包含了以下指令。

指令名称：redis2_query

语法：redis2_query cmd arg1 arg2 ...

默认值：no

使用环境：location，location if

功能：该指令用于指定一个带有参数的 redis 命令，工作方式类似于 redis-cli 工具。在单个 location 中可以多次使用该指令，这些查询将会按顺序执行。

例如：

```
location /pipelined {
    redis2_query set hello world;
    redis2_query get hello;

    redis2_pass 127.0.0.1:$TEST_NGINX_REDIS_PORT;
}
```

访问/pipelined 将会是以下的响应：

```
+OK
$5
world
```

指令名称：redis2_raw_query

语法：redis2_raw_query QUERY

默认值：no

使用环境：location，location if

功能：该指令用于指定原始的 redis 操作，它可以解释 Nginx 的变量，并将其作为它的操作参数。该指令只接受一个 redis 命令，如果想在单个查询中指定多个命令，那么可以使用 redis2_raw_queries 指令。

指令名称：redis2_raw_queries

语法：redis2_raw_queries N QUERIES

默认值：no

使用环境：location，location if

功能：该指令用于指定多个 redis 命令。第一个参数 N 用于指定命令的个数，而第二个参数则是具体的 redis 命令。这两个参数都可以使用 Nginx 变量。

例如：

```
location /pipelined {
redis2_raw_queries 3 "flushall\r\nget key1\r\nget key2\r\n";
redis2_pass 127.0.0.1:6379;
}

# GET /pipelined2?n=2&cmds=flushall%0D%0Aget%20key%0D%0A
location /pipelined2 {
set unescape uri $n $arg n;
set unescape uri $cmds $arg cmds;

redis2_raw_queries $n $cmds;

redis2 pass 127.0.0.1:6379;
}
```

注意：这个例子中的 set_unescape_uri 指令由 HttpSetMiscModule 提供。

指令名称：redis2_literal_raw_query

功能：该指令用于指定一个原始的 redis 查询，但是不能够使用 Nginx 变量。换句话说，在命令的参数中可以自由地使用$符号。该指令只允许使用一个 redis 指令。

语法：redis2_literal_raw_query QUERY

默认值：no

使用环境：location，location if

指令名称：redis2_pass

语法：redis2_pass <upstream_name> redis2_pass <host>:<port>

默认值：no

使用环境：location，location if

功能：该指令用于指定后台的 redis 服务器。

指令名称：redis2_connect_timeout

语法：redis2_connect_timeout <time>

默认值：60s

使用环境：http，server，location

功能：该指令用于设置连接后台数据库 redis 的超时时间，单位为秒。

指令名称：redis2_send_timeout

语法：redis2_send_timeout <time>

默认值：60s

使用环境：http，server，location

功能：该指令用于指定向 redis 服务器发送 TCP 请求的超时时间，单位为秒。

指令名称：redis2_read_timeout

语法：redis2_read_timeout <time>

默认值：60s

使用环境：http，server，location

功能：该指令用于设置从 redis 服务器读取 TCP 响应的超时时间，单位为秒。

指令名称：redis2_buffer_size

语法：redis2_buffer_size <size>

默认值：4k/8k

使用环境：http，server，location

功能：该指令用于设置缓存的大小。

指令名称：redis2_next_upstream

语法：redis2_next_upstream [error | timeout | invalid_response | off]

默认值：error timeout

使用环境：http，server，location

功能：该指令用于确定一个后台服务器失败的条件，从而使得这个请求被转到另一台后端的 redis 服务器。注意使用该指令需要 redis2_pass 指令设置了两个以上的 redis 后台服务器。

例如：

```
upstream redis_cluster {
server 127.0.0.1:6379;
server 127.0.0.1:6380;
}

server {
location /redis {
```

```
    redis2_next_upstream error timeout invalid_response;
    redis2_query get foo;
    redis2_pass redis_cluster;
    }
    }
```

3. 配置实例

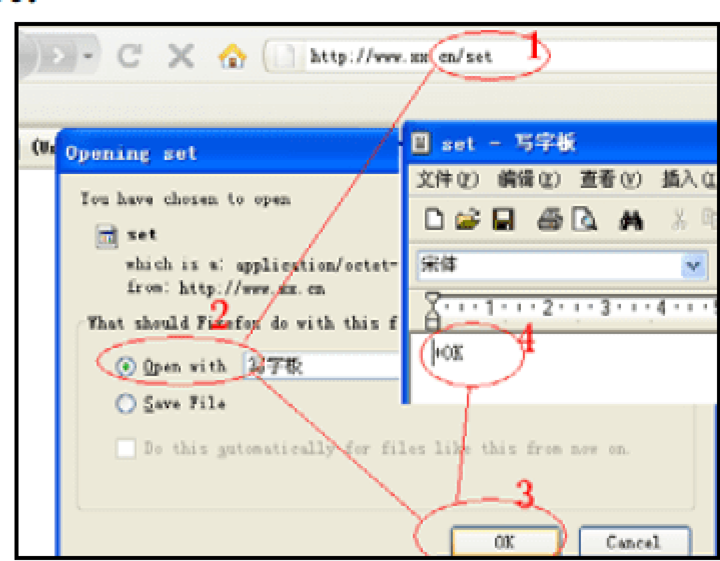
我们通过以下的一个例子简要说明它们的使用，在 Nginx 的配置文件中添加以下内容：

```
location /set {
    set $value 'first';
    redis2 query set c $value;
    redis2 pass 192.168.3.192:6379;
}

location /get {
redis2_query get c;
    redis2_pass 192.168.3.192:6379;
}
```

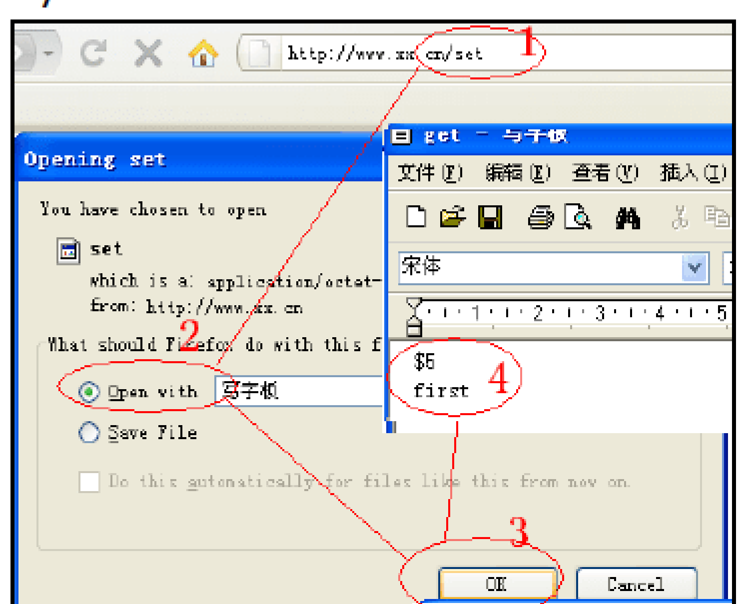
4. 访问测试

访问 http://www.xx.cn/set：



返回的响应内容为“+OK”。

继续访问 http://www.xx.cn/set：

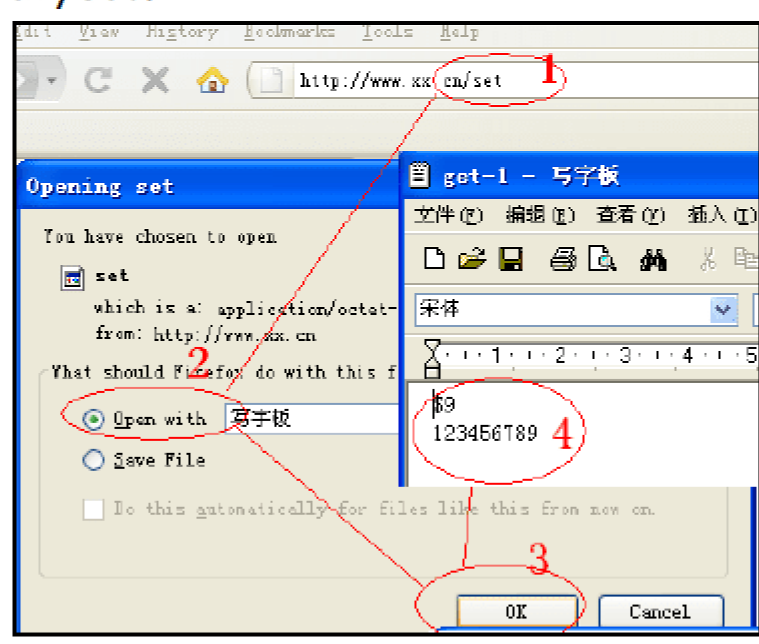


返回了变量的值，其中$5 表示变量值的大小，即 ASCII 的数量。

下面我们通过命令行做些改变：

```
[root@mfs2 src]# ./redis-cli
redis 127.0.0.1:6379> get c
"first"
redis 127.0.0.1:6379> set c 123456789
OK
redis 127.0.0.1:6379> get c.html
"123456789"
```

再次访问 http://www.xx.cn/set：

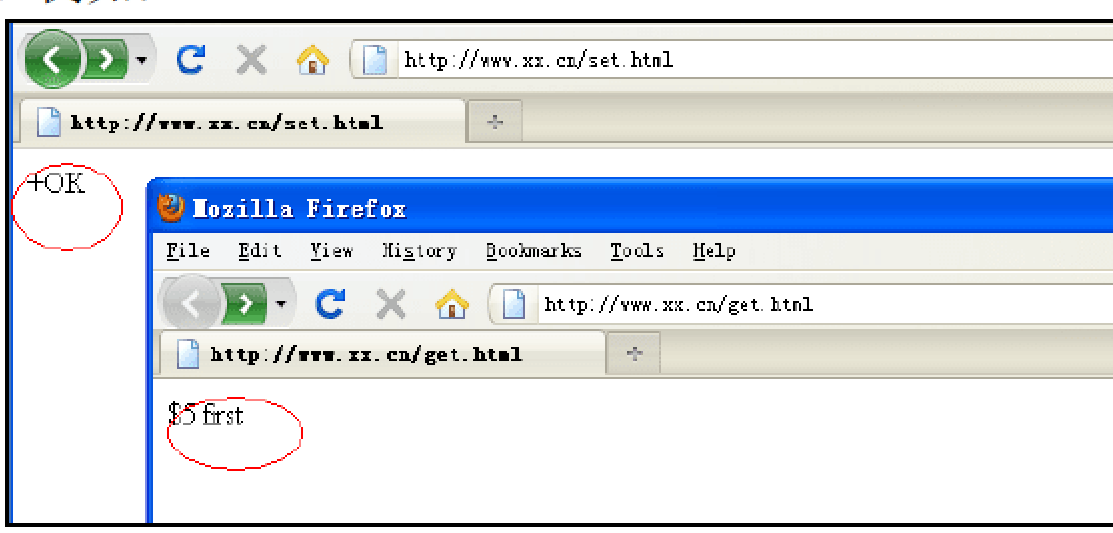


如果配置为以下方式：

```
location /set.html {
   set $value 'first';
   redis2 query set c $value;
   redis2 pass 192.168.3.192:6379;
}

location /get.html {
redis2_query get c;
   redis2_pass 192.168.3.192:6379;
}
```

那么访问就简单了。例如：



5. 支持 keep-live 功能

同 redis 一样，也可以使用 HttpUpstreamKeepaliveModule 来实现一个数据库连接池来提供 redis 的 TCP 连接。使用了 keep-live 的连接之后的配置可以使用如下配置：

```
http {
upstream backend {
```

```
    server 127.0.0.1:6379;

    # a pool with at most 1024 connections
    # and do not distinguish the servers:
    keepalive 1024 single;
  }

  server {
  …
  location /redis {
  set_unescape_uri $query $arg_query;
  redis2 query $query;
  redis2 pass backend;
  }
  }
  }
```

6. 与 Lua 协作

该模块能够为 HttpLuaModule 作为一个非阻塞工作方式的 redis2 客户端，例如下面的例子：

```
location /redis {
internal;

# set_unescape_uri is provided by ngx_set_misc
set_unescape_uri $query $arg_query;

redis2 raw query $query;
redis2 pass 127.0.0.1:6379;
}

location /main {
content_by_lua '
local res = ngx.location.capture ("/redis",
{ args = { query = "ping\\r\\n" } }
)
ngx.print ("[" .. res.body .. "]")
';
}
```

在这个例子中，使用了 GET 子请求。然后访问/main：

```
[+PONG\r\n]
```

这里的 \r\n 是一个 CRLF（Carriage-Return Line-Feed 回车换行），该模块从远程的 redis 服务器返回原始的 TCP 响应。对于基于 Lua 的应用程序开发者，他们可能想利用 LuaRedisParser 库（使用纯 C 编写）来解析这样的原始响应。就是说通过这个库将原始的 redis 响应解析为 Lua

数据结构。

50.3 关于 redis

redis 数据库也在悄然流行使用，它卓越的性能是被认可的首要条件。

1. redis 的功能

以下内容来源于互联网。

redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存，亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。从 2010 年 3 月 15 日起，redis 的开发工作由 VMware 主持。

作为 Key-Value 型数据库，redis 也提供了键（Key）和键值（Value）的映射关系。但是，除了常规的数值或字符串，redis 的键值还可以是以下形式之一：

- Lists （列表）
- Sets （集合）
- Sorted sets （有序集合）
- Hashes （哈希表）

键值的数据类型决定了该键值支持的操作。redis 支持诸如列表、集合或有序集合的交集、并集、差集等高级原子操作；同时，如果键值的类型是普通数字，redis 则提供自增等原子操作。

数据模型

通常，redis 将数据存储于内存中，或被配置为使用虚拟内存。通过两种方式可以实现数据持久化：使用快照的方式，将内存中的数据不断写入磁盘；或使用类似 MySQL 的日志方式，记录每次更新的日志。前者性能较高，但是可能会引起一定程度的数据丢失；后者相反。

支持主从数据库

redis 支持将数据同步到多台从库上，这种特性对提高读取性能非常有益。

性能

相比需要依赖磁盘记录每个更新的数据库，基于内存的特性无疑给 Redis 带来了非常优秀的性能。读写操作之间有显著的性能差异。

提供的 API

为以下语言提供 API：

- C
- C++
- C#
- Clojure
- Common Lisp
- Erlang

- Haskell
- Java
- Javascript
- Lua
- Objective-C
- Perl
- PHP
- Python
- Ruby
- Scala
- Go
- Tcl

2. redis 的安装

虽然这不是本书的内容，但是我们还是使用最少的篇幅讲述一下 redis 数据库的安装，谁让我们是做运维的！

下载并解压

```
[root@mfs2 ~]# http://redis.googlecode.com/files/redis-2.4.3.tar.gz
[root@mfs2 ~]# tar -zxvf redis-2.4.3.tar.gz
[root@mfs2 ~]# cd redis-2.4.3
[root@mfs2 redis-2.4.3]# tree -L 1
.
|-- 00-RELEASENOTES
|-- BUGS
|-- CONTRIBUTING
|-- COPYING
|-- INSTALL
|-- Makefile
|-- README
|-- TODO
|-- deps
|-- redis.conf
|-- runtest
|-- src
|-- tests
'-- utils

4 directories, 10 files
```

安装 redis

从解压包的帮助文件中看出，redis 的安装很有意思，在安装 redis 中既没有 configure 也没

有 make install 的操作，而是直接执行 make：

```
[root@mfs2 redis-2.4.3]# make
```

执行完成 make 命令，成功编译之后就可以直接使用了。使用方法如下：

```
[root@mfs2 src]# pwd
/root/redis-2.4.3/src
[root@mfs2 src]# ls -l -X
total 7412
…
-rwxr-xr-x  1 root root 1606182 Nov 23 19:35 redis-benchmark
-rwxr-xr-x  1 root root   14573 Nov 23 19:35 redis-check-aof
-rwxr-xr-x  1 root root   25939 Nov 23 19:35 redis-check-dump
-rwxr-xr-x  1 root root 1623075 Nov 23 19:35 redis-cli
-rwxr-xr-x  1 root root 2085781 Nov 23 19:35 redis-server

…
```

上面列出了生成的可执行命令，redis-server 命令用于执行服务器端运行，而 redis-cli 是一个客户端程序。

运行 redis 服务器

我们看一下 redis-server 命令的用法：

```
[root@mfs2 src]# ./redis-server  --help
Usage: ./redis-server [/path/to/redis.conf]
   ./redis-server - (read config from stdin)
```

该命令的用法比较简单，可以使用指定的配置文件，源代码中有配置文件的示例，也可以在命令行中添加配置选项，如果没有具体的设置则会使用默认的配置，例如：

```
[root@mfs2 src]# ./redis-server
[28405] 24 Nov 12:58:00 # Warning: no config file specified, using the default
config. In order to specify a config file use 'redis-server /path/to/redis.conf'
[28405] 24 Nov 12:58:00 * Server started, Redis version 2.4.3
[28405] 24 Nov 12:58:00 # WARNING overcommit_memory is set to 0! Background
save may fail under low memory condition. To fix this issue add
'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command
'sysctl vm.overcommit_memory=1' for this to take effect.
[28405] 24 Nov 12:58:00 * DB loaded from disk: 0 seconds
[28405] 24 Nov 12:58:00 * The server is now ready to accept connections on
port 6379
[28405] 24 Nov 12:58:00 - DB 0: 1 keys (0 volatile) in 4 slots HT.
[28405] 24 Nov 12:58:00 - 0 clients connected (0 slaves), 547512 bytes in
use
[28405] 24 Nov 12:58:05 - DB 0: 1 keys (0 volatile) in 4 slots HT.
[28405] 24 Nov 12:58:05 - 0 clients connected (0 slaves), 547512 bytes in
use
```

运行命令行客户端

我们看一下 redis-cli 命令的用法。

```
[root@mfs2 src]# ./redis-cli --help
redis-cli 2.4.3

Usage: redis-cli [OPTIONS] [cmd [arg [arg ...]]]
  -h <hostname>Server hostname (default: 127.0.0.1)
  -p <port>Server port (default: 6379)
  -s <socket> Server socket (overrides hostname and port)
  -a <password>Password to use when connecting to the server
  -r <repeat> Execute specified command N times
  -i <interval>When -r is used, waits <interval> seconds per command.
   It is possible to specify sub-second times like -i 0.1.
  -n <db> Database number
  -x  Read last argument from STDIN
  -d <delimiter>  Multi-bulk delimiter in for raw formatting (default: \n)
  --rawUse raw formatting for replies (default when STDOUT is not a tty)
  --latencyEnter a special mode continuously sampling latency.
  --help  Output this help and exit
  --versionOutput version and exit

Examples:
  cat /etc/passwd | redis-cli -x set mypasswd
  redis-cli get mypasswd
  redis-cli -r 100 lpush mylist x
  redis-cli -r 100 -i 1 info | grep used_memory_human:

When no command is given, redis-cli starts in interactive mode.
Type "help" in interactive mode for information on available commands.
```

该命令的用法基本上类似于 MySQL 数据库的客户端程序 mysql。如果没有使用具体的参数，那么将会使用默认的参数来连接 redis 数据库。例如：

```
[root@mfs2 src]# ./redis-cli
redis 127.0.0.1:6379>
```

下面是 redis 服务器报出的日志：

```
 [28415] 24 Nov 13:01:05 - Accepted 127.0.0.1:47314
```

测试操作

```
[root@mfs2 src]# ./redis-cli
redis 127.0.0.1:6379> set a 123
OK
redis 127.0.0.1:6379> get a
"123"
```

```
redis 127.0.0.1:6379>
```

程序部署

前面我们了解到，由于没有提供 make install，所以生成的可执行文件和源代码混杂在一起，而且可能也不是我们要运行的目录位置，因此需要我们手动将可执行文件和配置文件移动到一个能够按照我们的意志运行的目录中。例如：

```
[root@mfs2 src]# mkdir -p /usr/local/redis-2.4.3/bin
[root@mfs2 src]# mkdir -p /usr/local/redis-2.4.3/etc
[root@mfs2 src]# cp redis-benchmark redis-cli redis-server redis-check-aof \
redis-check-dump /usr/local/redis-2.4.3/bin
[root@mfs2 redis-2.4.3]# cp redis.conf  /usr/local/redis-2.4.3/etc/
```

另外，使用在生产环境中的 redis 一定要认真地设置它的配置文件。

redis 将所有的数据都放置在内存中，对于内存大小的使用和数据的安全性一定要考虑，因此使用主从数据库更加安全。

第 51 章　Nginx 访问 MongoDB

MongoDB 在网站中使用较多，主要使用它来存储静态文件，例如图像文件，因此在这一章我们将来认识一下 nginx-gridfs 模块，使用该模块来实现对 MongoDB 的访问，实际上就是 MongoDB 的客户端。在具体的安装中要注意它和数据库驱动的版本问题，有时候不是很好安装。

51.1 安装 nginx-gridfs 模块

下面我们首先来下载并且安装该模块。

下载 nginx-gridfs 模块：

```
[root@mail ~]# wget https://nodeload.github.com/mdirolf/nginx-gridfs
/tarball/master
--17:20:24--
https://nodeload.github.com/mdirolf/nginx-gridfs/tarball/master
Resolving nodeload.github.com... 207.97.227.252
Connecting to nodeload.github.com|207.97.227.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 19079 (19K) [application/octet-stream]
Saving to: 'mdirolf-nginx-gridfs-v0.8-11-ge5d8cc7.tar.gz'

100%[==================================>] 19,079  24.2K/s   in 0.8s

17:20:28 (24.2 KB/s) - 'mdirolf-nginx-gridfs-v0.8-11-ge5d8cc7.tar.gz' saved
[19079/19079]
```

解压下载包：

```
[root@mail ~]# tar -zxvf mdirolf-nginx-gridfs-v0.8-11-ge5d8cc7.tar.gz
```

下载 mongodb 驱动：

```
[root@mfs2 ~]# wget https://nodeload.github.com/mongodb/mongo-c-driver
/tarball/master
--19:37:12--
https://nodeload.github.com/mongodb/mongo-c-driver/tarball/master
   => 'master'
Resolving nodeload.github.com... 207.97.227.252
Connecting to nodeload.github.com[207.97.227.252]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 77,331 [application/octet-stream]

100%[================================>] 77,33157.55K/s
```

```
19:37:19 (57.47 KB/s) - 'master' saved [77,331/77,331]
```

错误的保存为 master，因此需要重命名：

```
[root@mfs2 ~]# mv master mongodb-mongo-c-driver-v0.4-17-g68aa48e.tar.gz
```

解压数据包：

```
[root@mfs2 ~]# tar -zxvf mongodb-mongo-c-driver-v0.4-17-g68aa48e.tar.gz
```

将 mongodb 数据库驱动移动到 nginx-gridfs 模块下面的 mongo-c-driver/目录：

```
[root@mfs2 mongodb-mongo-c-driver-68aa48e]# mv  ./* /root/   \
> mdirolf-nginx-gridfs-e5d8cc7/mongo-c-driver/
```

编译安装：

```
[root@mfs2 nginx-1.0.10]# ./configure --prefix=/usr/local/nginx-1.0.10
-gridfs  \
> --add-module=/root/mdirolf-nginx-gridfs-e5d8cc7
```

1. 配置示例

例 1

```
location /gridfs/ {
gridfs my_app;
}
```

例 2

```
location /gridfs/ {
gridfs my app field=filename type=string;
mongo 127.0.0.1:27017;
}
```

例 3

```
location /gridfs/ {
gridfs my app field=filename type=string;
mongo "foo"
  10.7.2.27:27017
  10.7.2.28:27017;
}
```

例 4

```
location /gridfs/ {
gridfs my_app
   root_collection=pics
   field=_id
   type=int
   user=foo
   pass=bar;
mongo 127.0.0.1:27017;
}
```

2. 指令

分析其源代码：

```
static ngx command t ngx http gridfs commands[] = {

{
ngx_string ("mongo") ,
NGX_HTTP_LOC_CONF | NGX_CONF_1MORE,
ngx_http_mongo,
NGX HTTP LOC CONF OFFSET,
0,
NULL
},

{
ngx_string ("gridfs") ,
NGX_HTTP_LOC_CONF | NGX_CONF_1MORE,
ngx http gridfs,
NGX HTTP LOC CONF OFFSET,
0,
NULL
},

ngx_null_command
};
```

可以看出提供了以下两条指令。

指令名称：gridfs

语法：gridfs DB_NAME [root_collection=ROOT] [field=QUERY_FIELD] [type=QUERY_TYPE] [user=USERNAME] [pass=PASSWORD]

默认值：NONE

使用环境：location

功能：该指令用于在一个给定的 location 中启用 nginx-gridfs 模块，仅有一个必须的参数，那就是 DB_NAME，用于指定提供数据（即文件）的数据库。

可选项如下。

- root_collection：指定 GridFS 的 root_collection（即前缀），默认值为 fs。
- field：指定查询的字段，支持的字段包括_id and 文件名，默认值为_id。
- type：指定查询的类型，支持的类型有 objectid、string 和 int，默认值为 objectid。
- user：指定连接 mongo 数据库的用户，如果 mongo 数据库使用了用户认证，那么会用到这个参数，默认值为空。
- pass：指定连接 mongo 数据的用户密码，如果 mongo 数据库设置了用户认证，那么会用到这个参数，默认值为空。

指令名称：mongo

语法：mongoMONGOD_HOSTmongoREPLICA_SET_NAME MONGOD_SEED_1 MONGOD_SEED_2

默认值：127.0.0.1:27017

使用环境：location

功能：该指令用于指定一个连接的 mongod 或者是 replica 组（就是设置一个组名称）。MONGOD_HOST 的格式应该是 hostname:port，REPLICA_SET_NAME 应该是要连接的 replica 组的名称。如果没有该指令，那么该模块将会尝试连接运行在 127.0.0.1:27017 套接字下的 MongoDB。

3. 配置实例

```
location /pics/ {

gridfs pics
  field=filename
  type=string;
mongo 192.168.3.157:27017;
}
```

4. 访问测试

我们在浏览器中输入/pics/目录的文件就可以访问到该文件。在此就不再截图了。

51.2 关于 MongoDB

在互联网中，MongoDB 的应用已经成熟，而且使用得也比较多，MongoDB 是一个基于分布式文件存储的数据库，它是由 C++语言编写的。开发 MongoDB 的目的就是为 Web 应用提供可扩展的高性能的数据存储方案，它的特点是高性能、易部署、易使用，存储数据非常方便。

1. MongoDB 的功能

> 下面的描述来自于互联网。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库中功能最丰富，最像关系数据库的。它支持的数据结构非常松散，类似 json 的 bjson 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

MongoDB 的主要功能特性如下。

- 面向集合存储，易存储对象类型的数据。
- 模式自由。
- 支持动态查询。
- 支持完全索引，包含内部对象。

- 支持查询。
- 支持复制和故障恢复。
- 使用高效的二进制数据存储，包括大型对象（如视频等）。
- 自动处理碎片，以支持云计算层次的扩展性。
- 支持 Ruby、Python、Java、C++、PHP 等多种语言。
- 文件存储格式为 BSON（一种 JSON 的扩展）。
- 可通过网络访问。

所谓“面向集合”（Collenction-Orented），意思是数据被分组存储在数据集中，被称为一个集合（Collenction）。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库（RDBMS）里的表（table），不同的是它不需要定义任何模式（schema）。

模式自由（schema-free），意味着对于存储在 MongoDB 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库中。存储在集合中的文档，被存储为键—值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为 BSON（Binary Serialized dOcument Format）。

MongoDB 服务端可运行在 Linux、Windows 或 OS X 平台，支持 32 位和 64 位应用，默认端口为 27017。推荐运行在 64 位平台，因为 MongoDB 在 32 位模式运行时支持的最大文件尺寸为 2GB。

MongoDB 将数据存储在文件中（默认路径为：/data/db），为提高效率使用内存映射文件进行管理。

2. MongoDB 的安装

虽然这不是本书的内容，但我们还是使用最少的篇幅讲述一下 MongoDB 数据库的安装，谁让我们是做运维的！

下载并解压

```
    [root@mail ~]# wget http://fastdl.mongodb.org/linux/mongodb-linux-i686
-2.0.1.tgz
    --16:39:54--
http://fastdl.mongodb.org/linux/mongodb-linux-i686-2.0.1.tgz
    Resolvingfastdl.mongodb.org...216.137.55.190,216.137.55.203, 216.137.
55.247, ...
    Connecting to fastdl.mongodb.org|216.137.55.190|:80... connected.
    HTTP request sent, awaiting response... 200 OK
    Length: 38067066 (36M) [application/x-tar]
    Saving to: 'mongodb-linux-i686-2.0.1.tgz'

    100%[================>] 38,067,066  23.6K/s   in 19m 53s

    16:59:50(31.2KB/s)-'mongodb-linux-i686-2.0.1.tgz'saved[38067066/38067066]
```

```
[root@mail ~]# tar -zxvf mongodb-linux-i686-2.0.1.tgz
[root@mail ~]# cd mongodb-linux-i686-2.0.1/
[root@mail mongodb-linux-i686-2.0.1]# tree
.
|-- GNU-AGPL-3.0
|-- README
|-- THIRD-PARTY-NOTICES
'-- bin
|-- bsondump
|-- mongo
|-- mongod
|-- mongodump
|-- mongoexport
|-- mongofiles
|-- mongoimport
|-- mongorestore
|-- mongos
|-- mongosniff
|-- mongostat
'-- mongotop

1 directory, 15 files
```

在这个目录结构中，README 文件是一个帮助文件，也是我们的入口文件，通过它我们可以了解到 MongoDB 的大致用法。

其中，mongod 是数据库的运行进程，它的命令行参数很多，如果没有具体的设置，直接运行该命令就可以了，但是在默认使用中需要 “dbpath=/data/db/” 的设置。因此首先需要指定该目录，当然也可以通过“--dbpath”选项来指定具体的目录。

例如：

```
[root@mail bin]# ./mongod
./mongod --help for help and startup options
Fri Nov 25 14:46:02
Fri Nov 25 14:46:02 warning: 32-bit servers don't have journaling enabled
by default. Please use --journal if you want durability.
Fri Nov 25 14:46:02
Fri Nov 25 14:46:02 [initandlisten] MongoDB starting : pid=31510 port=27017
dbpath=/data/db/ 32-bit host=mail
Fri Nov 25 14:46:02 [initandlisten]
Fri Nov 25 14:46:02 [initandlisten] ** NOTE: when using MongoDB 32 bit, you
are limited to about 2 gigabytes of data
Fri Nov2514:46:02[initandlisten]**see http://blog.mongodb.org/post/1377
88967/32-bit-limitations
Fri Nov 25 14:46:02 [initandlisten] **  with --journal, the limit is lower
```

```
Fri Nov 25 14:46:02 [initandlisten]
Fri Nov 25 14:46:02 [initandlisten] db version v2.0.1, pdfile version 4.5
Fri Nov 2514:46:02[initandlisten] git version:3a5cf0e2134a830d38d2d1aae7e
88cac31bdd684
Fri Nov 25 14:46:02 [initandlisten] build info: Linux domU-12-31-39-01-70-
B42.6.21.7-2.fc8xen #1 SMP Fri Feb 1512:39:36 EST 2008 i686 BOOST LIB
 VERSION=1 41
Fri Nov 25 14:46:02 [initandlisten] options: {}
Fri Nov 25 14:46:02 [websvr] admin web console waiting for connections on
port 28017
Fri Nov 25 14:46:02 [initandlisten] waiting for connections on port 27017
```

而 mongo 是一个客户端，类似于 Mysqld 的客户端 mysql。例如：

```
[root@mail bin]# ./mongo
MongoDB shell version: 2.0.1
connecting to: test
> use pics;
switched to db pics
> show dbs
local   (empty)
pics0.0625GB
test0.0625GB
> help
db.help () help on db methods
db.mycoll.help ()  help on collection methods
rs.help () help on replica set methods
help admin   administrative help
help connect connecting to a db help
help keyskey shortcuts
help miscmisc things to know
help mr  mapreduce

show dbs show database names
show collections show collections in current database
show users show users in current database
show profile   show most recent system.profile entries with time >= 1ms
show logsshow the accessible logger names
show log [name]  prints out the last segment of log in memory, 'global' is
default
use <db_name>set current database
db.foo.find () list objects in collection foo
db.foo.find ( { a : 1 } ) list objects in foo where a == 1
it  result of the last line evaluated; use to further iterate
DBQuery.shellBatchSize = x   set default number of items to display on shell
```

```
exit
> use collections (pics{filename:80,type:string})
switched to db collections (pics{filename:80,type:string})
>  show dbs
collections (pics{filename:80,type:string})   (empty)
local    (empty)
pics0.0625GB
test0.0625GB
```

在上面的执行中，我们创建了数据库 pics。

其他的命令和工具就不再介绍了，使用起来还是比较容易的。

3. 访问测试

首先我们进行以下操作，即创建数据库和添加访问文件：

添加数据库：

```
[root@mail bin]# pwd
/root/mongodb-linux-i686-2.0.1/bin
[root@mail bin]# ./mongo
MongoDB shell version: 2.0.1
connecting to: test
> use pics
switched to db pics
>
```

添加访问文件：

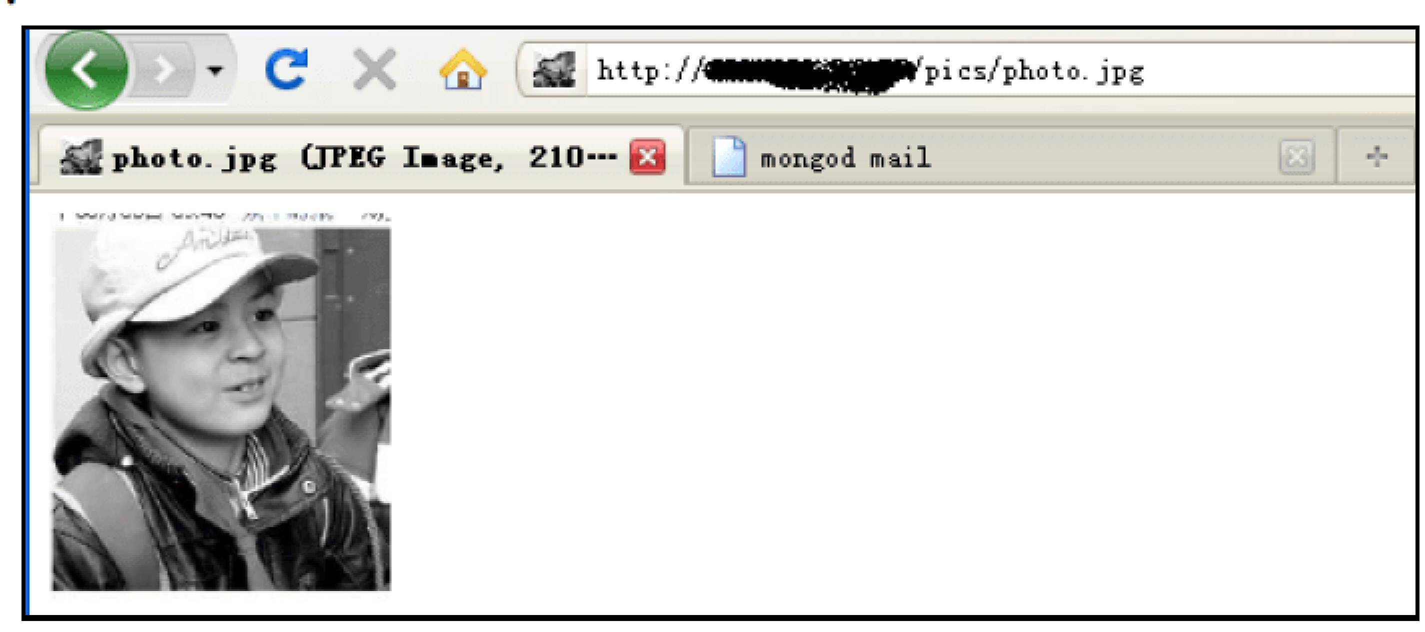
```
[root@mail bin]# ./mongofiles --db pics put  index.html
[root@mail bin]# ./mongofiles --db pics put  1.jpg
[root@mail bin]# ./mongofiles --db pics put  2.txt
[root@mail bin]# ./mongofiles --db pics put  photo.jpg
[root@mail bin]# ./mongofiles --db pics list
connected to: 127.0.0.1
photo.jpg      11652
1.jpg   25727
2.txt   14
1.jpg   25727
index.html     144
```

访问数据库中的文件：

文本文件：

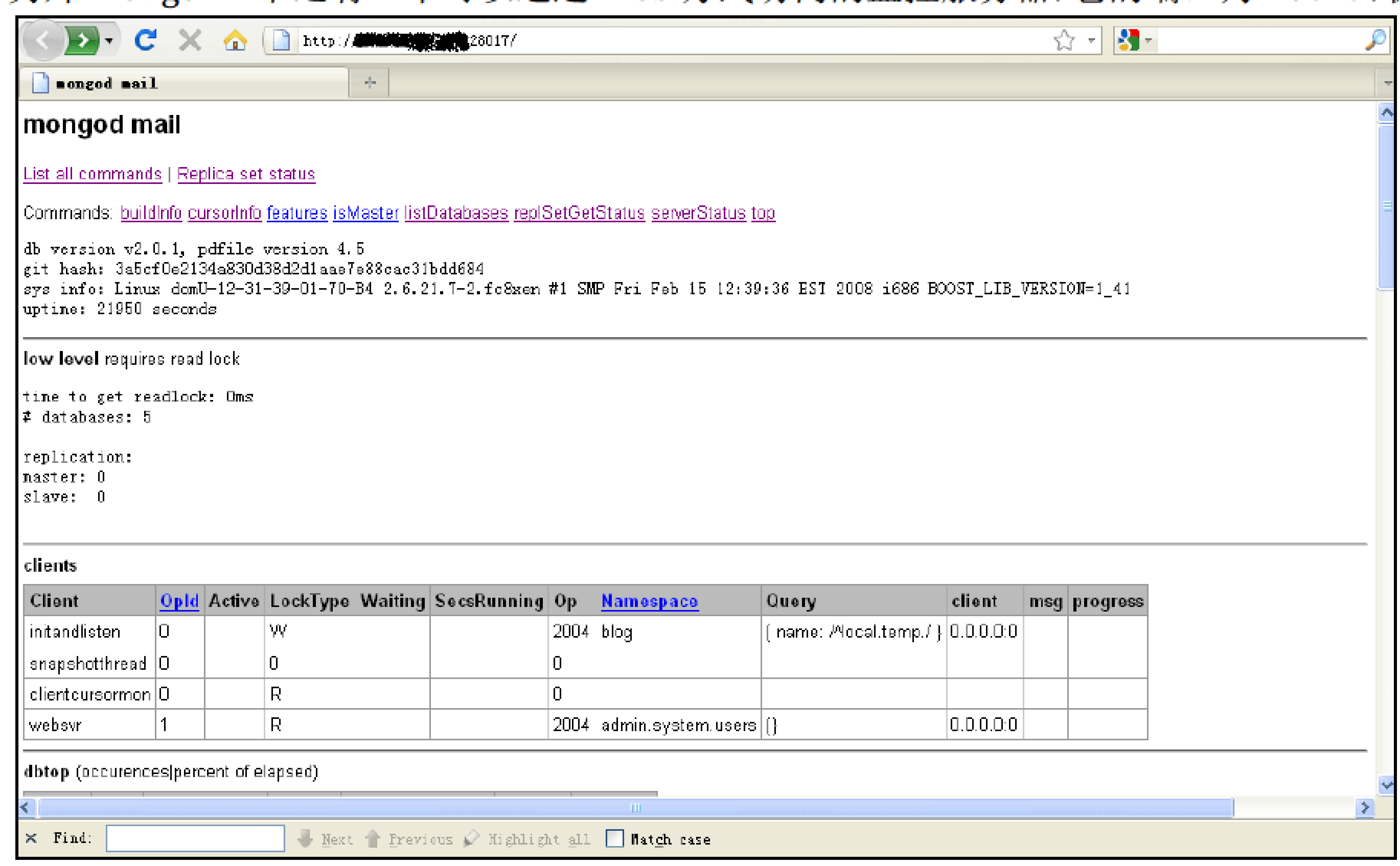


图像文件：



其他的命令和工具就不再介绍了，使用起来还是比较容易。

另外 MongoDB 下还有一个可以通过 Web 方式访问的监控服务器，它的端口为 28017，例如：



第 52 章　Nginx 访问 Mogilefs

Mogilefs 在互联网中使用也比较广泛，因此这一章我们来认识一下 mogilefs_module 模块。

1. 安装 mogilefs_module 模块

下载 nginx_mogilefs 模块：

```
[root@mail ~]# wget http://www.grid.net.ru/nginx/download  \
> /nginx_mogilefs_module-1.0.4.tar.gz
```

解压并查看目录结构：

```
[root@mail ~]# tar -zxvf nginx_mogilefs_module-1.0.4.tar.gz
[root@mail ~]# cd nginx_mogilefs_module-1.0.4
[root@mail nginx_mogilefs_module-1.0.4]# tree
.
|-- Changelog
|-- LICENCE
|-- README
|-- config
|-- nginx.conf
'-- ngx_http_mogilefs_module.c

0 directories, 6 files
```

README 文件中没有什么内容，相关的内容可以查看 http://www.grid.net.ru/nginx/mogilefs.en.html，文件 nginx.conf 是一个示例性配置文件。

安装 nginx_mogilefs 模块：

```
[root@mail nginx-1.0.8]# ./configure --prefix=/usr/local  \
/nginx-1.0.8-mogilefs
> --add-module=/root/nginx_mogilefs_module-1.0.4
[root@mail nginx-1.0.8]# make
[root@mail nginx-1.0.8]# make install
```

2. 配置示例

```
server {
listen   80;

location /download/ {
#
# Query tracker at 192.168.2.2 for a file with the key
# equal to remaining part of request URI
#
```

```
    mogilefs_tracker 192.168.2.2;

    mogilefs_pass {
    proxy pass $mogilefs path;
    proxy hide header Content-Type;
    proxy buffering off;
    }
    }
    }
```

3. 指令

我们看一下它的源代码：

```
    tatic ngx command t  ngx http mogilefs commands[] = {

    { ngx_string("mogilefs_pass"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_NOARGS|NGX
 CONF TAKE1|NGX CONF BLOCK,
      ngx http mogilefs pass block,
      NGX HTTP LOC CONF OFFSET,
      0,
      NULL },

    { ngx_string("mogilefs_tracker"),

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx http mogilefs tracker command,
      NGX_HTTP_LOC_CONF_OFFSET,
      0,
      NULL },

    { ngx_string("mogilefs_domain"),

NGX HTTP MAIN CONF|NGX HTTP SRV CONF|NGX HTTP LOC CONF|NGX CONF TAKE1,
      ngx conf set str slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, domain),
      NULL },

    { ngx_string("mogilefs_connect_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
```

```
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, upstream.connect_timeout),
      NULL },

    { ngx_string("mogilefs_send_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, upstream.send_timeout),
      NULL },

    { ngx_string("mogilefs_read_timeout"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_msec_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, upstream.read_timeout),
      NULL },

    { ngx_string("mogilefs_noverify"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_TAKE1,
      ngx_conf_set_flag_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, noverify),
      NULL },

    { ngx_string("mogilefs_methods"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_conf_set_bitmask_slot,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, methods),
      &ngx_http_mogilefs_methods_mask },

    { ngx_string("mogilefs_class"),

NGX_HTTP_MAIN_CONF|NGX_HTTP_SRV_CONF|NGX_HTTP_LOC_CONF|NGX_CONF_1MORE,
      ngx_http_mogilefs_class_command,
      NGX_HTTP_LOC_CONF_OFFSET,
      offsetof(ngx_http_mogilefs_loc_conf_t, class_templates),
```

```
    NULL },

    ngx_null_command
  };
```

可以看出，以上源代码中有以下几条指令。

指令名称：mogilefs_pass

语法：mogilefs_pass [<key>] {<fetch block>}

默认值：none

使用情况：强制使用

使用环境：server，location，limit_except

功能：该指令用于指定从 MogileFS tracker 上查询文件的 key。key 中可以包含任何变量，如果忽略 key，那么请求中 URI 部分将会被作为 key 来使用，当然包括与 location 中匹配的部分：

```
  location /download/ {
   mogilefs pass {
   [...]
   }
  }
```

在这个例子中，如果客户端请求中的 URI 是/download/example.jpg，那么 tracker 查询文件的 key 为 example.jpg。

“fetch block”包含了用于 Nginx 获取文件的存储节点，在这个区段中必须包含使用带有$mogilefs_path 参数的 proxy_pass 指令。例如：

```
  mogilefs_pass {
     proxy_pass $mogilefs_path;
     proxy hide header Content-Type;
     proxy buffering off;
  }
```

变量$mogilefs_path 包含了存储在节点上文件的绝对 URL，“fetch block”创建了一个隐蔽的，提供内部访问的 location，它的名字为/mogilefs_fetch_XXXXXXXX，在 tracker 成功地响应之后将会执行重定向。

变量$mogilefs_path1 .. $mogilefs_path9 包含了在备用存储节点文件的 URL。

指令名称：mogilefs_methods

语法：mogilefs_methods <[[method 1] method 2 ...]>

默认值：GET

使用环境：main，server，location

使用情况：可选

功能：该指令用于指定什么样的指令可以访问 MogileFS，GET 指令用于从 MogileFS 中获取一个资源，而 PUT 指令用于创建或者是替换一个资源，DELETE 指令可以从 MogileFS 中删除一个资源。

指令名称：mogilefs_domain

语法：mogilefs_domain <domain>

默认值：default

使用环境：main，server，location

使用情况：可选

功能：该指令用于指定在 MogileFS 中查询的范围（domain），可以包含变量。

指令名称：mogilefs_class

语法：mogilefs_class <class0> [<class1> [...]]

默认值：N/A

使用环境：main，server，location

使用情况：可选

功能：该指令用于指定向 tracker 产生一个请求时使用什么样的“class”作为参数。该指令参数将会被评估，并且第一个不为空的值将会被作为 class 使用，参数可以包含变量。

指令名称：mogilefs_tracker

语法：mogilefs_tracker <IP|IP:port|upstream>

默认值：none

使用环境：main，server，location

使用情况：强制使用

功能：该指令用于指定用于查询 MogileFS tracker 的 IP 地址。

指令名称：mogilefs_noverify

语法：mogilefs_noverify <on/off>

默认值：off

使用环境：main，server，location

使用情况：可选

功能：该指令用于启用向 MogileFS 发送不用校验的参数。

指令名称：mogilefs_connect_timeout

语法：mogilefs_connect_timeout <time>

默认值：60s

使用环境：main，server，location

使用情况：可选

功能：该指令用于指定连接 mogilefs tracker 的超时时间，这个值不能大于 45 秒。

指令名称：mogilefs_send_timeout

语法：mogilefs_send_timeout <time>

默认值：60s

使用环境：main，server，location

使用情况：可选

功能：该指令用于指定向 mogilefs tracker 发送数据的超时时间，如果在这个时间间隔内 mogilefs tracker 没有接收到数据，那么 Nginx 将会关闭连接。

指令名称：mogilefs_read_timeout

语法：mogilefs_read_timeout <time>

默认值：60s

使用环境：main，server，location

使用情况：可选

功能：该指令用于设置从 mogilefs tracker 上读取数据，换句话说就是从 mogilefs tracker 上获取数据的超时时间，如果 Nginx 在这个指定的时间间隔内没有收到 mogilefs tracker 发送回的数据，那么 Nginx 将会关闭这个连接。

4. 配置实例

```
server {

listen 80;
…

location ~ ^/pic/ {

rewrite / (.*)  $1 break;
mogilefs tracker 192.168.3.159:6001;
mogilefs domain pic.xx.com;
mogilefs_methods get;
mogilefs_pass {
proxy_pass $mogilefs_path;
proxy_hide_header Content-Type;
proxy buffering off;
}

…

}

}
```

5. 访问测试

由于涉及内容较多，在此就不再进行测试了。

6. 关于 MogileFS

MogileFS 是一套高效的文件自动备份组件，由 Six Apart 开发，广泛应用在包括 LiveJournal 等 Web 2.0 站点上。

MogileFS 由 3 部分组成。

第一部分是server端，包括mogilefsd和mogstored两个程序。前者即是mogilefsd的tracker，它将一些全局信息保存在数据库中，例如站点 domain、class、host 等；后者即是存储节点（store node），它其实是一个 HTTP Daemon，默认侦听在 7500 端口，接受客户端的文件备份请求。在安装完后，要运行 mogadm 工具将所有的 store node 注册到 mogilefsd 的数据库中，mogilefsd 会对这些节点进行管理和监控。

第二部分是 utils（工具集），主要是 MogileFS 的一些管理工具，例如 mogadm 等。

第三部分是客户端 API，目前只有 Perl API（MogileFS.pm），用这个模块可以编写客户端程序，实现文件的备份管理功能。

—— 来自于互联网

第 3 部分

Nginx 与缓存

通过 Nginx 来实现缓存功能基本上有三类方法，其中：

第一类方法 Nginx 自带，即 proxy_cache、proxy_store 和 memcached，在没有 proxy_cache 之前只能用 proxy_store 缓存页面，但是 memcached 需要第三方的软件 Memcached；

第二类是通过添加第三方模块；

第三类是通过 Varnish 服务器。

这样，其实共有五种方法来实现缓存，下面我们通过例子来分析。

第 53 章　缓存技术——proxy_cache

Nginx 的这个功能从 0.7.48 版本开始提供的，它开始支持类似 Squid 的缓存功能。它的原理是把 URL 及相关变量的组合当做 Key，再用 MD5 编码，编码后的哈希值作为文件名，然后再将缓存的文件保存在硬盘中。我们现在使用的是 0.8.53 版本，早在 0.8.31 版本中，proxy_cache 就比较完善了。之所以说它比较完善，其实就是说它没有缓存清除机制，是通过第三方的模块 ngx_cache_purge 来清除指定的 URL 缓存，它支持任意 URL 链接、支持 404/301/302/200 状态码缓存，因此，在使用反向代理的同时使用 Nginx 的 proxy_cache 缓存机制是个很不错的做法。

53.1　了解 cache_purge 模块

编译安装 Nginx，这次安装的主要操作在于添加 ngx_cache_purge 这个第三方的模块，操作如下：

```
    [root@cache ~]# wget http://labs.frickle.com/files/ngx cache purge
-1.2.tar.gz
    [root@cache ~]# tar -xvzf  ngx cache purge-1.2.tar.gz
    ngx_cache_purge-1.2
    ngx_cache_purge-1.2/t
    ngx_cache_purge-1.2/t/proxy.t
    ngx_cache_purge-1.2/ngx_cache_purge_module.c
    ngx cache purge-1.2/config
    ngx cache purge-1.2/README
    ngx cache purge-1.2/LICENSE
    ngx_cache_purge-1.2/CHANGES
    [root@cache ~]#cd  ngx_cache_purge-1.2
    [root@cache ~]# wget http://www.nginx.org/download/nginx-0.8.53.tar.gz
    [root@cache ~]# tar -xvzf nginx-0.8.53.tar.gz
    [root@cache ~]#cd nginx-0.8.53
    [root@cache nginx-0.8.53]./configure --prefix=/usr/local/nginx0.8.53 \
    --add-module=../ngx cache purge-1.2
    [root@cache nginx-0.8.53] make
    [root@cache nginx-0.8.53] make install
```

进入 ngx_cache_purge-1.2 目录，查看一下 README 文件：

```
    [root@cache ~]#cd  ngx cache purge-1.2
    [root@cache ngx cache purge-1.2]cat README
    ABOUT:
    ------
    ngx_cache_purge is nginx module which adds ability to purge content
    from FastCGI, proxy, SCGI and uWSGI caches.
```

```
SPONSORS:
---------
Work on the original patch was fully funded by yo.se.

......
```

它的具体内容就不全贴出来了，在这里我们来分析一下 README 文件，在该文件中，它主要说明了两点，一点是该模块提供的指令的用法，该模块提供了如下 4 条指令。

指令及使用环境	描　述
fastcgi_cache_purge 使用环境：location	设置区域（area），换句话就是说通过 fastcgi_cache 指令指定的 cache 区域（zone） 语法格式： fastcgi_cache_purge zone_name key 其中 zone_name 为指令 fastcgi_cache 指定的 cache 区域（zone）；而参数 key 就是指令 fastcgi_cache_key 的参数
proxy_cache_purge 使用环境：location	设置区域（area），换句话就是说通过 proxy_cache 指令指定的 cache 区域（zone） 语法格式： proxy_cache_purge zone_name key 其中 zone_name 为指令 proxy_cache 指定的 cache 区域（zone）；而参数 key 就是指令 proxy_cache_key 的参数
scgi_cache_purge 使用环境：location	大概意思同上。 语法格式： scgi_cache_purge zone_name key
uwsgi_cache_purge 使用环境：location	大概意思同上。 语法格式： uwsgi_cache_purge zone_name key

另一点就是提供了一个配置示例：

```
http {
proxy cache path  /tmp/cache  keys zone=tmpcache:10m;

server {
location / {
proxy_pass http://127.0.0.1:8000;
proxy_cachetmpcache;
proxy_cache_key$uri$is_args$args;
}

location ~ /purge (/.*)  {
allow  127.0.0.1;
deny   all;
proxy_cache_purge  tmpcache  $1$is_args$args;
```

```
    }
  }
}
```

要留意一下上面配置中的黑体字部分。

53.2 设置 Nginx 的配置文件

看下面的配置文件：

```
[root@cache conf]# cat nginx.conf
…
http {
…
 client_body_buffer_size  512k;
 proxy_connect_timeout5;
 proxy_read_timeout   60;
 proxy_send_timeout   5;
 proxy_buffer_size16k;
 proxy_buffers4 64k;
 proxy_busy_buffers_size  128k;
 proxy_temp_file_write_size 128k;
 proxy_temp_path   /usr/local/nginx0.8.53/proxy_temp;
proxy_cache_path /usr/local/nginx0.8.53/proxy_cache levels=1:2 keys_zone
=content:20m inactive=1d max_size=100m;

   server {
listen   80;
server name  localhost;
location / {
root   html;
index  index.html index.htm;
}

   …

location ~* \.jsp$ {
proxy_cache content;
proxy_cache_valid  200 304 301 302 10d;
proxy_cache_valid  any 1d;
proxy_cache_key $host$uri$is_args$args;
proxy_passhttp://192.168.3.139:8080;
}
```

```
      location ~ /purge (/.*)  {
          allow  192.168.3.0/24;
          deny   all;
          proxy_cache_purge  content  $host$1$is_args$args;

     location ~* \. (gif|jpg|jpeg|png|bmp|swf|js|html|htm|css) $ {
  proxy_cache content;
  proxy_cache_valid  200 304 301 302 10d;
  proxy_cache_valid  any 1d;
  proxy_set_header Host  $host;
          proxy_set_header X-Forwarded-For  $remote_addr;
  proxy_cache_key$host$uri$is_args$args;
  proxy_pass http://192.168.3.139:8080;
     }
  }

  …
```

在这个配置文件中，我们看以下四个部分。

第一部分：设定缓存。

```
 client body buffer size  512k;
 proxy connect timeout5;
 proxy_read_timeout   60;
 proxy_send_timeout   5;
 proxy_buffer_size16k;
 proxy_buffers4 64k;
 proxy busy buffers size  128k;
 proxy temp file write size 128k;
 proxy temp path   /usr/local/nginx0.8.53/proxy temp;
 proxy cache path /usr/local/nginx0.8.53/proxy cache levels=1:2 keys zone
=content:20m inactive=1d max_size=100m;
```

第二部分：缓存 jsp 文件。

```
location ~* \.jsp$ {
proxy cache content;
proxy cache valid  200 304 301 302 10d;
proxy_cache_valid  any 1d;
proxy_cache_key $host$uri$is_args$args;
proxy_passhttp://192.168.3.139:8080;
}
```

第三部分：清除缓存。

```
 location ~ /purge (/.*)  {
        allow  192.168.3.0/24;
        deny   all;
        proxy_cache_purge  content  $host$1$is_args$args;
```

第四部分：缓存静态文件。

```
    location ~* \.(gif|jpg|jpeg|png|bmp|swf|js|html|htm|css) $ {
proxy_cache content;
proxy_cache_valid  200 304 301 302 10d;
proxy_cache_valid  any 1d;
proxy_set_header Host  $host;
    proxy_set_header X-Forwarded-For  $remote_addr;
proxy_cache_key$host$uri$is_args$args;
proxy_pass http://192.168.3.139:8080;
  }
```

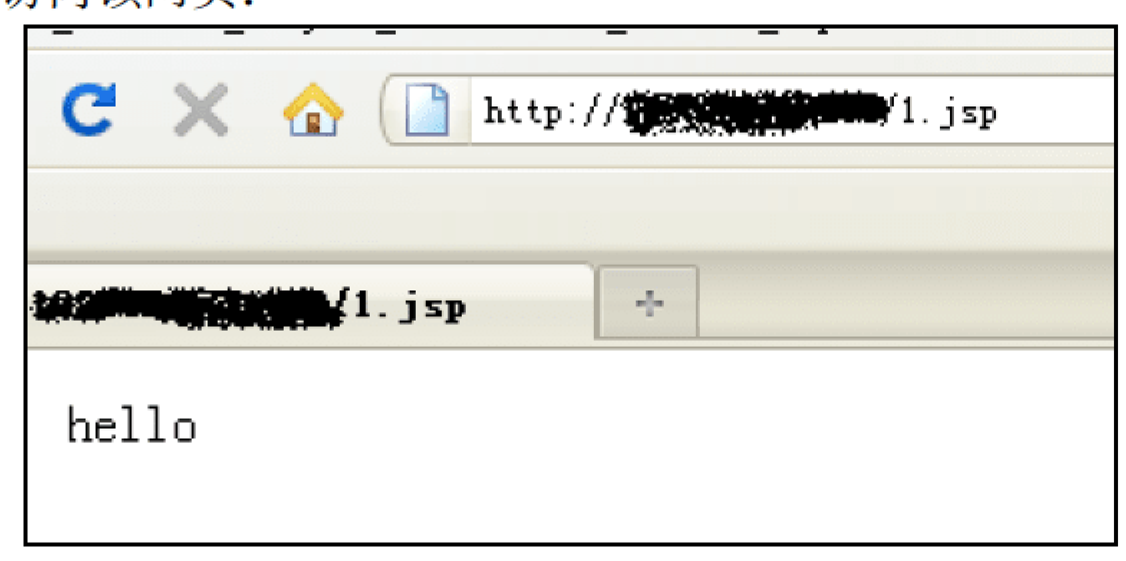
在上面的配置中，我们对.jsp 文件进行了缓存。在具体的使用中要根据情况而定，一般不会对动态的文件进行缓存。另外，静态文件一般也不会走 Tomcat，不过如果真的走了 Tomcat 了，现在不是也有缓存了！之所以要分开讨论这个四个部分，就是想更清楚地认识一下缓存策略，对于缓存策略我们各自有不同的需求，因此也就不能一概而论了。

53.3 访问测试

在 Tomcat 的 ROOT 目录下建立一个 jsp 文件，内容如下：

```
[root@cache ROOT]# pwd
/usr/local/tomcat/webapps/ROOT
[root@cache ROOT]# more 1.jsp
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<%
out.print("hello");
%>
```

然后通过浏览器访问该网页：



同时查看 Tomcat 的访问日志以及 Nginx 的访问日志：

```
[root@cache logs]pwd
/usr/local/tomcat/logs
[root@cache local]# tail -f localhost_access_log.2010-12-29.txt
…
192.168.3.139 -- [29/Dec/2010:16:40:27 +0800] "GET /1.jsp HTTP/1.0" 200 7
```

```
    [root@cache ~]# tail -f /usr/local/nginx0.8.53/logs/access.log
    …
    192.168.3.248 - - [29/Dec/2010:16:40:27 +0800] "GET /1.jsp HTTP/1.1" 200
7 "-" "Mozilla/5.0 ( Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12 )
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

可以看得出，在访问 1.jsp2 时，在 Tomcat 及 Nginx 均有访问日志输出。

然后再次做同样的请求，同时查看日志输出：

在本次的访问中，Tomcat 中访问日志没有输出，而 Nginx 的日志输出如下：

```
    192.168.3.248 - - [29/Dec/2010:16:44:44 +0800] "GET /1.jsp HTTP/1.1" 200
7 "-" "Mozilla/5.0 ( Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12 )
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

> 同时还要注意，http 的返回代码为 200，在后面我们将进行一个访问图片的例子，注意对比一下。

然后进入缓存目录查看被缓存的文件：

```
    [root@cache 8c]# pwd
    /usr/local/nginx0.8.53/proxy cache/9/8c
    [root@cache 8c]# more 876c6e39b8a5d8c9b4cb49c1bdf848c9
    ? (M→M2?e7
    KEY: 192.168.3.139/1.jsp
    HTTP/1.1 200 OK
    Server: Apache-Coyote/1.1
    Content-Type: text/html;charset=utf-8
    Content-Length: 7
    Date: Wed, 29 Dec 2010 08:17:23 GMT
    Connection: close



    Hello
```

可以明确的看出，它是以静态化的形式存储，这种形式就是被人们称为的“伪静态”网页。

下面的例子是访问 http://192.168.3.139/index.jsp，即 Tomcat 的默认主页，网页中有 8 个元素，其中 7 个为静态文件，一个为.jsp。

下面是 Tomcat 的日志输出，它记录了 8 个元素的访问情况，均为 200 状态：

```
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /index.jsp HTTP/1.0"
200 12684
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /tomcat.css HTTP/1.0"
200 6065
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /asf-logo.png HTTP/1.0"
200 17811
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /tomcat.png HTTP/1.0"
200 5103
```

```
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /bg-nav.png HTTP/1.0"
200 1401
    192.168.3.139 - - [30/Dec/2010:08:59:05 +0800] "GET /bg-upper.png HTTP/1.0"
200 3103
    192.168.3.139  -  -  [30/Dec/2010:08:59:05  +0800]  "GET  /bg-button.png
HTTP/1.0" 200 713
    192.168.3.139  -  -  [30/Dec/2010:08:59:05  +0800]  "GET  /bg-middle.png
HTTP/1.0" 200 1918
```

下面是 Nginx 的日志输出，同样它记录了 8 个元素的访问情况，均为 200 状态：

```
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /index.jsp HTTP/1.1" 200
12697 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /tomcat.css HTTP/1.1"
200 6065 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /asf-logo.png HTTP/1.1"
200 17811 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /tomcat.png HTTP/1.1"
200 5103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /bg-nav.png HTTP/1.1"
200 1401 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111 - - [30/Dec/2010:08:59:05 +0800] "GET /bg-upper.png HTTP/1.1"
200 3103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111  -  -  [30/Dec/2010:08:59:05  +0800]  "GET  /bg-button.png
HTTP/1.1" 200 713 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; GTB6.6)"
    192.168.3.111  -  -  [30/Dec/2010:08:59:05  +0800]  "GET  /bg-middle.png
HTTP/1.1" 200 1918 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; GTB6.6)"
```

将客户端的缓存清除后，再次访问同样的网页，然后与上次的访问比较一下。

下面是清除了浏览器缓存后再次访问的结果：

下面是 Tomcat 的日志输出，它只是再次被访问了 index.jsp，其他的静态元素均未被再次拉出：

```
    192.168.3.139 - - [30/Dec/2010:09:00:53 +0800] "GET /index.jsp HTTP/1.0"
200 12684
```

而 Nginx 的日志就不一样了，它和第一次的访问一样，均被重新拉出了一次，8 个元素一个没少：

```
    192.168.3.111 - - [30/Dec/2010:09:00:53 +0800] "GET /index.jsp HTTP/1.1" 200
12697 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6)"
```

```
192.168.3.111 - - [30/Dec/2010:09:00:53 +0800] "GET /tomcat.css HTTP/1.1" 200 6065 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:53 +0800] "GET /asf-logo.png HTTP/1.1" 200 17811 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:53 +0800] "GET /tomcat.png HTTP/1.1" 200 5103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:53 +0800] "GET /bg-nav.png HTTP/1.1" 200 1401 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:54 +0800] "GET /bg-button.png HTTP/1.1" 200 713 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:54 +0800] "GET /bg-upper.png HTTP/1.1" 200 3103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
192.168.3.111 - - [30/Dec/2010:09:00:54 +0800] "GET /bg-middle.png HTTP/1.1" 200 1918 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6) "
```

如果我们把缓存目录下的所有文件和目录全部删除再次访问，那么记录的结果将会和第一次访问完全一样。在这里就不浪费篇幅了：

```
[root@cache proxy cache]# pwd
/usr/local/nginx0.8.53/proxy_cache
[root@cache proxy_cache]# tree
.
|-- 1
|   '-- 59
|   '-- 82e087ad6aefcb2bebbeee5c0f6a8591
|-- 4
|   |-- 98
|   |   '-- 9cf99950f077a247f146d07566a3e984
|   '-- bf
|   '-- a753cb470bec6f583e1a572027743bf4
|-- 5
|   '-- 3a
|   '-- d90a32d908b6bd3023b2474936a943a5
|-- 8
|   '-- 67
|   '-- 4cabf27755faee5167a26e893e432678
|-- 9
|   |-- 0e
```

```
    |    |    '-- d21ca01e33a31f841a5feade425040e9
    |    '-- 8c
    |    '-- 876c6e39b8a5d8c9b4cb49c1bdf848c9
    '-- e
    '-- f7
    '-- 02acdbd50567b6230f294b5ba530bf7e

14 directories, 8 files
[root@cache proxy_cache]#
[root@cache proxy_cache]# rm -fR *
```

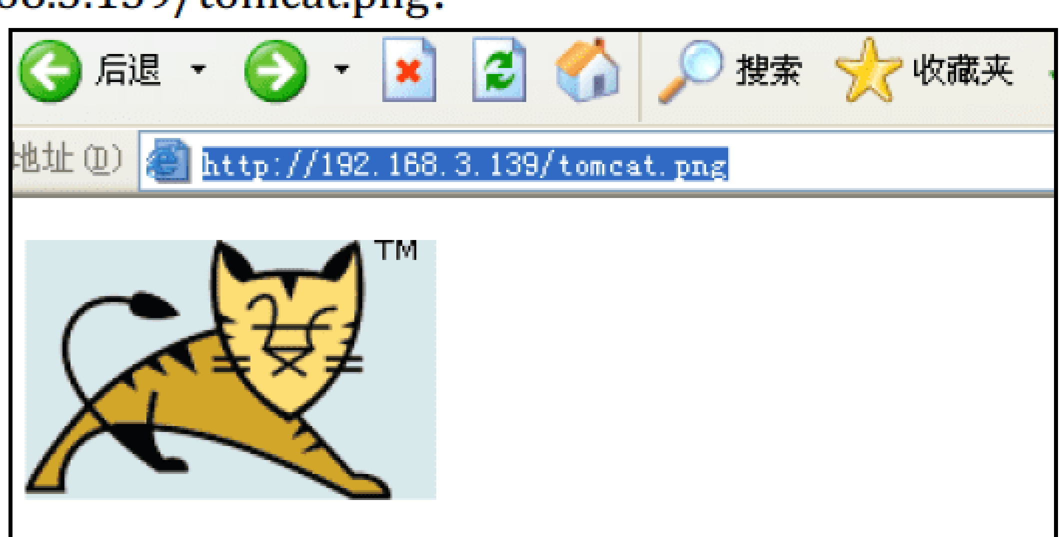
作为一个总结，你可能注意到所有目录下的文件名均为一个没有意义的字母数字组合。没错，如果你再仔细点还会发现所有的文件名均为 32 位，这就是将 URL 及相关变量的组合当做 Key，再用 MD5 哈希编码的结果。

53.4 手动清除缓存

下面的一个例子是针对清除指定的 URL 缓存。在编译安装 Nginx 的时候使用了第三方模块 ngx_cache_purge，该模块的功能就是清除指定的 URL 缓存。

我们首先访问一个图片文件，然后查看缓存目录。

访问 http://192.168.3.139/tomcat.png：



查看缓存目录。在下面的代码中，第一次使用 tree 命令是在访问前进行的，而第二个 tree 命令是在访问之后进行的：

```
[root@cache proxy_cache]# tree
.
'-- 4
'-- bf

2 directories, 0 files
[root@cache proxy_cache]# ###以上是访问前的结构，以下是访问后的结构#####
[root@cache proxy_cache]# tree
.
'-- 4
'-- bf
'-- a753cb470bec6f583e1a572027743bf4
```

```
2 directories, 1 file
```

清除缓存。清除缓存的方法就是在访问的路径中加入 purge。对于上面的访问地址来说就是 http://192.168.3.139/purge/tomcat.png：

址(D) http://192.168.3.139/purge/asf-logo.png

Successful purge

Key : 192.168.3.139/asf-logo.png
Path: /usr/local/nginx0.8.53/proxy_cache/c/f7/02acdbd50567b6230f294b5ba530bf7c

nginx/0.8.53

然后再查看缓存目录情况：

```
[root@cache proxy_cache]# tree
.
'-- 4
'-- bf

2 directories, 0 files
```

没错已被清除了。

最后一点需要注意的是 Nginx 的进程情况：

```
[root@cache modules]# ps -ef|grep nginx
Root 32157 1 0 0 8:21 ? 00:00:00 nginx: master process /usr/local/nginx
-purge/sbin/nginx
nobody  32158 32157  0 08:21 ?00:00:00 nginx: worker process
nobody  32159 32157  0 08:21 ?00:00:00 nginx: cache manager process
nobody  32160 32157  0 08:21 ?00:00:00 nginx: cache loader process
```

在原有的基础上多出了两个进程，这就是管理缓存的文件的进程，其中 cache loader 在 Nginx 启动一会儿后会自动退出。

第 54 章　缓存技术——proxy_store

在没有 proxy_cache 之前，都是使用 proxy_store，由于 Nginx 的 proxy_store 技术当时在设计时没有采用任何刷新机制，因此，在缓存的管理上也就更人为了一些（你不觉得这么说更好听些？！），我们可以自己写个脚本进行缓存清理，等等。使用 proxy_store 的原理其实也很简单，那就是 Nginx 首先在本地查找客户端请求的内容，如果找不到就去 proxy_pass 指定的后端服务器上查找，然后会被保存到本地的缓存中。

> 使用 proxy_store 技术有一个缺点，其实也可以说是优点：它会在本地缓存中构建一个和远程服务器——proxy_pass 指定的后端服务器目录结构完全一样的结构（真的可以叫镜像了！），这是从它的优点方面讲，如果从缺点方面讲，那么它会浪费很大的磁盘空间，如果没有一个足够大的硬盘或者是没有及时地清除缓存中的过时文件，那么将是一个可怕的问题，所以你要么准备一个足够大的硬盘。要么及时或是有计划地清除缓存。

54.1 设置 Nginx 的配置文件

看一下配置情况：

```
…
http {
include   mime.types;
default_type  application/octet-stream;
sendfileon;
keepalive timeout  65;

   server {
listen   80;
server_name  localhost;
location / {
   root   html;
   index  index.html index.htm;
}

    location ~* \.jsp$ {
proxy_passhttp://192.168.3.139:8080;
   }

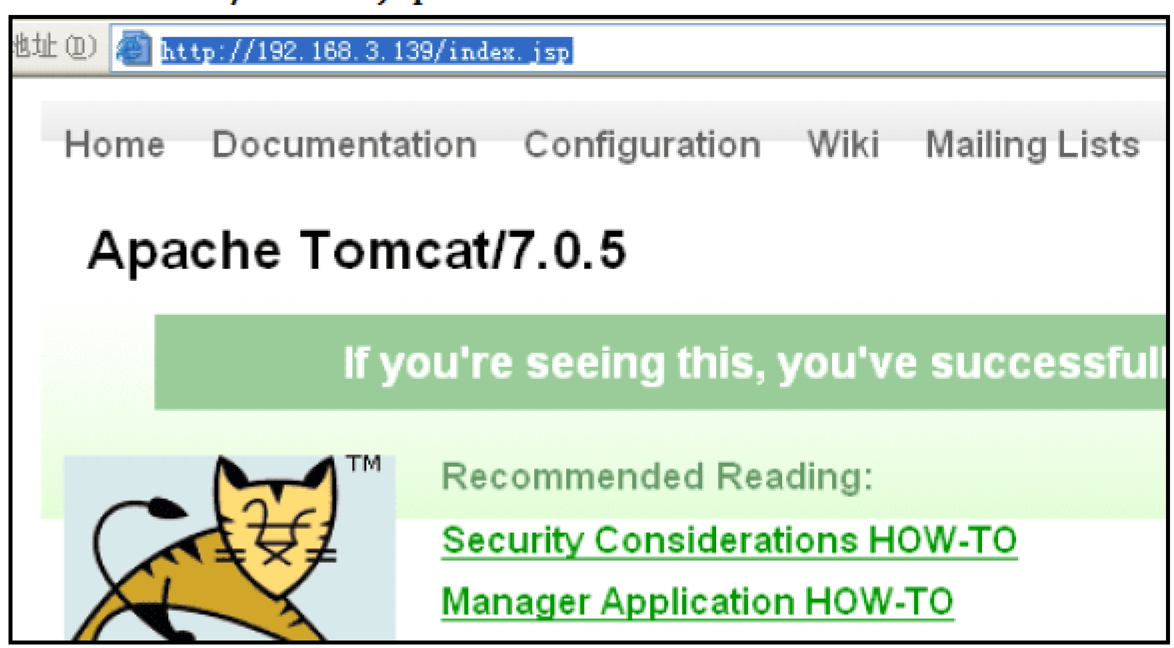
    location ~* \.(gif|jpg|jpeg|png|bmp|swf|js|html|htm|css)$ {
expires 3d;
```

```
proxy_set_header Accept-Encoding '';
root /usr/local/nginx0.8.53/proxy_temp;
proxy_store on;
proxy_store_access user:rw group:rw all:rw;
proxy_temp_path /usr/local/nginx0.8.53/proxy_temp;
if ( !-e $request_filename) {
proxy_passhttp://192.168.3.139:8080;
   }
}
```

注意配置中的黑体字部分，我们对这些类文件 gif、jpg、jpeg、png、bmp、swf、js、html、htm、css 做了缓存。

54.2 访问测试

访问 http://192.168.3.139/index.jsp：



然后查看缓存目录情况：

```
[root@cache proxy_temp]# tree
.

0 directories, 0 files
[root@cache proxy_ temp]# ###以上是访问前的结构，以下是访问后的结构####
[root@cache proxy temp]# tree
.
|-- asf-logo.png
|-- bg-button.png
|-- bg-middle.png
|-- bg-nav.png
|-- bg-upper.png
|-- tomcat.css
'-- tomcat.png
```

从缓存目录中可以看出，有 7 个元素被缓存。查看其访问日志。

下面是 Tomcat 的访问日志，有 8 个 GET 操作成功执行：

```
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /index.jsp HTTP/1.0"
200 12684
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /tomcat.css HTTP/1.0"
200 6065
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /asf-logo.png HTTP/1.0"
200 17811
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /tomcat.png HTTP/1.0"
200 5103
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-upper.png HTTP/1.0"
200 -
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-nav.png HTTP/1.0"
200 -
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-button.png
HTTP/1.0" 200 -
    192.168.3.139 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-middle.png
HTTP/1.0" 200 -
```

在 Nginx 的访问日志中同样有 8 个 GET 操作成功执行：

```
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /index.jsp HTTP/1.1"
200 12697 "-" "Mozilla/4.0 ( compatible; MSIE 6.0; Windows NT 5.1; SV1;
TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30; CIBA) "
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /tomcat.css HTTP/1.1"
200 6065 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /asf-logo.png HTTP/1.1"
200 17811 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /tomcat.png HTTP/1.1"
200 5103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-upper.png HTTP/1.1"
200 0 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
    192.168.3.248 - - [30/Dec/2010:18:48:50 +0800] "GET /bg-nav.png HTTP/1.1"
200 0 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA) "
```

```
    192.168.3.248  -  -  [30/Dec/2010:18:48:50  +0800]  "GET  /bg-button.png
HTTP/1.1" 200 0 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248  -  -  [30/Dec/2010:18:48:50  +0800]  "GET  /bg-middle.png
HTTP/1.1" 200 0 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
```

然后将浏览器的缓存清除后再进行一次访问，查看 Nginx 的访问日志和 Tomcat 的访问日志。

在 Tomcat 的访问日志中，有 1 个 GET 操作成功执行，那就是对 **index.jsp** 的获取：

```
    192.168.3.139 - - [30/Dec/2010:19:36:22 +0800] "GET /index.jsp HTTP/1.0"
200 12684
```

而在 Nginx 的访问日志中，仍有 8 个 GET 操作成功执行：

```
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /index.jsp HTTP/1.1"
200 12697 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30; CIBA)"
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /tomcat.css HTTP/1.1"
200 6065 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /asf-logo.png HTTP/1.1"
200 17811 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /tomcat.png HTTP/1.1"
200 5103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /bg-upper.png HTTP/1.1"
200 3103 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248 - - [30/Dec/2010:19:36:22 +0800] "GET /bg-nav.png HTTP/1.1"
200 1401 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible; MSIE 6.0;
Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248  -  -  [30/Dec/2010:19:36:22  +0800]  "GET  /bg-button.png
HTTP/1.1" 200 713 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
    192.168.3.248  -  -  [30/Dec/2010:19:36:22  +0800]  "GET  /bg-middle.png
HTTP/1.1" 200 1918 "http://192.168.3.139/index.jsp" "Mozilla/4.0 (compatible;
```

```
MSIE 6.0; Windows NT 5.1; SV1; TencentTraveler 4.0; .NET CLR 1.1.4322; .NET
CLR 2.0.50727; .NET CLR 3.0.04506.30; CIBA)"
```

如果你还想进一步地证实被缓存文件的访问情况，那么你可以使用 stat 命令查看它们被访问的时间（Access）。例如：

```
[root@cache proxy_temp]# stat tomcat.png
  File: 'tomcat.png'
  Size: 5103Blocks: 16  IO Block: 4096   regular file
Device: fd00h/64768dInode: 15158299Links: 1
Access: (0666/-rw-rw-rw-)  Uid: ( 99/ nobody)   Gid: (  99/  nobody)
Access: 2010-12-30 19:36:22.000000000 +0800
Modify: 2010-11-25 01:59:41.000000000 +0800
Change: 2010-12-30 18:48:50.000000000 +0800
```

54.3 手动清除缓存

接下来的一个问题就是需要解决清除缓存。做个定时清除缓存不是什么难事，首先编写一个 shell 脚本（这个脚本简单的不能再简单了），然后让它定时执行就可以了。

首先编辑脚本文件，然后保存并给予执行的权限：

```
[root@cache ~]# vi clear-cache.sh
#!/bin/sh
rm -fR  /usr/local/nginx0.8.53/proxy temp/*
[root@cache ~]
[root@cache ~]chmod clear-cache.sh
```

再添加定时执行任务：

```
[root@cache ~]# crontab -e

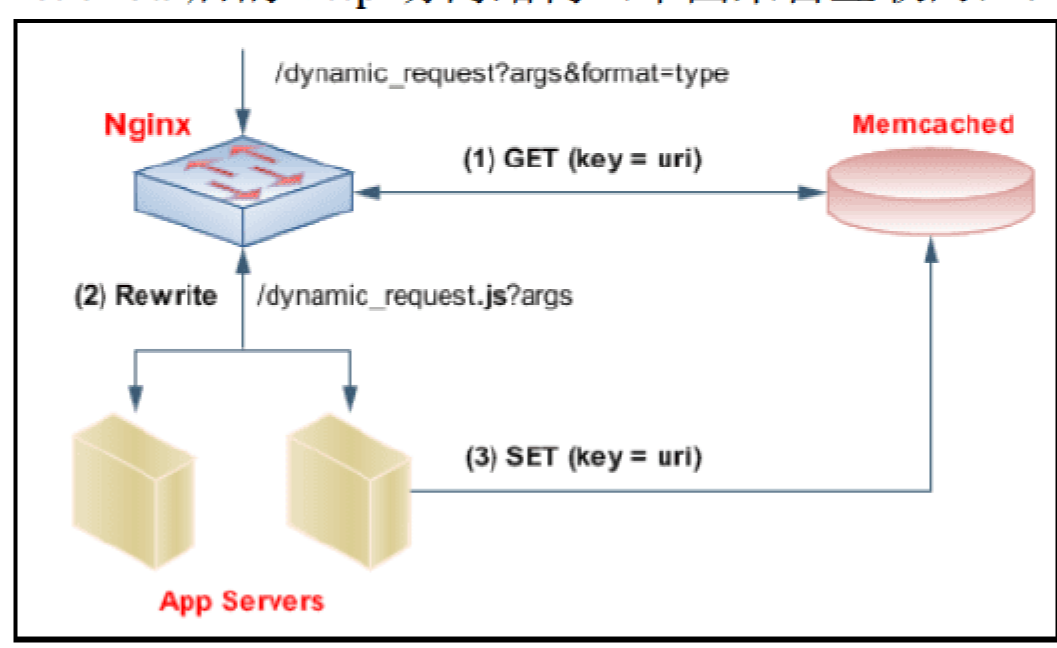
* * */3 * * /root/ clear-cache.sh
```

在这里我们指定了 3 天清除一次缓存。这个间隔根据实际情况而定，对于内容变化大的网站来说可以设定的间隔短一点，而对于内容变换不大的网站来说可以设定的间隔长一点。

第 55 章　缓存技术——Memcached

Memcached 模块也分为四部分来分析，第一部分是 Nginx 的 Memcached 模块本身，第二部分是 Memcached 服务，第三部分是 Nginx 的配置文件，第四部分是客户端。

下图是使用了 Memcached 后的 http 访问结构（本图来自互联网）：



55.1 Memcached 服务器

在这里我们先安装一下 Memcached。由于 Memcached 要使用 libevent 库，用它做 Socket 的处理，因此，我们需要安装 libevent 库，libevent 库的最新版本是 libevent-2.0.10-stable。使用 Memcached 模块需要注意，Nginx 目前还没有向后台服务 Memcached 写入的任何机制，因此要往 Memcached 服务中写入数据需要用后台的动态语言完成，也可以利用 404 定向到后端去写入数据。

看下面的这个图：

What is Memcached?

Free & open source, high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load.

Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Memcached is simple yet powerful. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Its API is available for most popular languages.

Memcached Users

- LiveJournal
- Wikipedia
- Flickr
- Bebo
- Twitter
- Typepad
- Yellowbot
- Youtube
- Digg
- WordPress.com
- Craigslist
- Mixi

Download Memcached

The latest stable memcached release is

v1.4.7

release notes (2011-8-16)

tar.gz

Source and Development

之所以截这个图是想说明都有哪些网站在使用 Memcached 服务器。

55.2 下载并安装 libevent 库

libevent 的官方网站：http://www.monkey.org/~provos/libevent/。

正如其名字，libevent 就是一个程序库，通过使用 libevent 库能够将 Linux 的 epoll、BSD 操作系统的 kqueue 等事件处理功能封装成统一的接口，并且在高负载请求的 Memcached 服务器下，一样能够发挥 O（1）的性能。Memcached 使用 libevent 库，因此能够在 Linux、BSD 操作系统上发挥其高性能。

libevent 是一个异步事件处理软件函数库，以 BSD 许可证释出。

libevent 提供了一组应用程序编程接口（API），让程序设计师可以设定某些事件发生时所执行的函数，也就是说，libevent 可以用来取代网络服务器所使用的事件循环检查架构。

由于可以省去对网络的处理，且拥有不错的效能，有些软件使用 libevent 作为网络底层的函数库，如：Memcached、Tor 等。

支持程度

目前 libevent 支持以下方式判断事件的发生：

* poll（2）

* select（2）

几乎所有的 UNIX 平台都有提供的函数。

* /dev/pool

以 Solaris 平台为主。

* kqueue（2）

以 BSD 平台为主。

* epoll（2）

以 Linux 平台为主。

—— 来源于互联网

1. 下载 libevent

```
[root@cache ~]# wget http://cdnetworks-kr-2.dl.sourceforge.net /\
> project/levent/libevent/libevent-2.0/libevent-2.0.10-stable.tar.gz
[root@cache ~]# tar -zxvf  libevent-2.0.10-stable.tar.gz
[root@cache libevent-2.0.10-stable]# cd libevent-2.0.10-stable
```

2. 安装 libevent

```
[root@cache libevent-2.0.10-stable]# ./configure --prefix=/usr/\
> local/libevent-2.0.10
[root@cache libevent-2.0.10-stable]#make
[root@cache libevent-2.0.10-stable]#make install
  …
----------------------------------------------------------------------
Libraries have been installed in:
   /usr/local/libevent-2.0.10/lib

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
 during execution
```

```
  - add LIBDIR to the 'LD_RUN_PATH' environment variable
 during linking
  - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
  - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------

…
```

注意我们安装的位置，查看安装情况：

```
[root@cache ~]# tree /usr/local/libevent-2.0.10/
/usr/local/libevent-2.0.10/
|-- bin
|   '-- event_rpcgen.py
|-- include
|   |-- evdns.h
|   |-- event.h
|   |-- event2
…
|   '-- evutil.h
'-- lib
|-- libevent-2.0.so.5 -> libevent-2.0.so.5.0.1
|-- libevent-2.0.so.5.0.1
|-- libevent.a
|-- libevent.la
|-- libevent.so -> libevent-2.0.so.5.0.1
|-- libevent_core-2.0.so.5 -> libevent_core-2.0.so.5.0.1
|-- libevent_core-2.0.so.5.0.1
|-- libevent_core.a
|-- libevent_core.la
|-- libevent_core.so -> libevent_core-2.0.so.5.0.1
|-- libevent_extra-2.0.so.5 -> libevent_extra-2.0.so.5.0.1
|-- libevent_extra-2.0.so.5.0.1
|-- libevent_extra.a
|-- libevent_extra.la
|-- libevent_extra.so -> libevent_extra-2.0.so.5.0.1
|-- libevent_openssl-2.0.so.5 -> libevent_openssl-2.0.so.5.0.1
|-- libevent_openssl-2.0.so.5.0.1
|-- libevent_openssl.a
|-- libevent_openssl.la
|-- libevent_openssl.so -> libevent_openssl-2.0.so.5.0.1
```

```
    |-- libevent_pthreads-2.0.so.5 -> libevent_pthreads-2.0.so.5.0.1
    |-- libevent_pthreads-2.0.so.5.0.1
    |-- libevent_pthreads.a
    |-- libevent_pthreads.la
    |-- libevent_pthreads.so -> libevent_pthreads-2.0.so.5.0.1
    '-- pkgconfig
    |-- libevent.pc
    |-- libevent_openssl.pc
    '-- libevent_pthreads.pc

5 directories, 59 files
```

由于我们在安装 libevent 库时没有安装在标准的 Linux 目录库下，因此需要在 ld.so.conf 文件中添加以下内容，然后执行 ldconfig 命令将库载入：

```
[root@cache bin]# vi /etc/ld.so.conf
…
/usr/local/libevent-2.0.10/lib/
[root@cache bin]# ldconfig
```

否则在执行 Memcached 启动时会出现“./memcached: error while loading shared libraries: libevent-2.0.so.5: cannot open shared object file: No such file or directory”这样的错误。

55.3 下载并安装 Memcached

这里 Memcached 的安装主要讲的是服务器端的安装，目前最新的版本是 Memcached-1.4.5。使用 Memcached 作为高速运行的缓存服务器，具有以下的特点：协议简单、基于 libevent 的事件处理和内存存储方式。

1. 安装 Memcached

从官方网站 http://memcached.org/下载最新的 Memcached 安装包。

```
[root@cache ~]# wget http://memcached.googlecode/
> .com/files/memcached-1.4.5.tar.gz
[root@cache ~]# tar -zxvf memcached-1.4.5.tar.gz
[root@cache ~]# cd memcached-1.4.5
[root@cache memcached-1.4.5]#
```

对于 Memcached 的安装，有必要先看一下它的 configure 脚本，或者至少应看一下它的帮助信息。例如：

```
[root@cache memcached-1.4.5]# ./configure -h
```

而且要特别注意以下部分：

```
Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE   do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
```

```
    --disable-dependency-tracking  speeds up one-time build
    --enable-dependency-tracking   do not reject slow dependency extractors
    --enable-sasl   Enable SASL authentication
    --enable-sasl-pwdb  Enable plaintext password db
    --enable-dtrace Enable dtrace probes
    --disable-coverage  Disable code coverage
    --enable-64bit  build 64bit version
    --disable-docs  Disable documentation generation
```

例如，如果我们想要开启 SASL 认证功能，那么就必须使用“--enable-sasl”选项，等等。安装步骤如下：

```
    [root@cache ~]# wget http://memcached.googlecode.com/files/memcached
-1.4.5.tar.gz
    [root@cache ~]# tar -zxvf memcached-1.4.5.tar.gz
    [root@cache memcached-1.4.5]# ./configure
    > --prefix=/usr/local/memcached-1.4.5
    > --with-libevent=/usr/local/libevent-2.0.10/
    [root@cache memcached-1.4.5]#make
    [root@cache memcached-1.4.5]#make install
```

在使用 Memcached 之前，我们先了解一下 Memcached 协议，如果你的英语不错，那么你可以直接去读 http://code.sixapart.com/svn/memcached/trunk/server/doc/protocol.txt，在这里只是对该内容的翻译和理解。

2. 可能出现的问题

在运行 Memcached 时出现问题：

```
    [root@nas ~]# memcached -h
    memcached: error while loading shared libraries: libevent-2.0.so.5: cannot
open shared object file: No such file or directory
    [root@nas ~]# whereis memcached
    memcached: /usr/local/bin/memcached
```

上面的错误说明是由于Memcached在启动的过程中去查找libevent-2.0.so.5库文件而没有找到，这种情况多数是属于路径找错的原因，因此我们通过调试模式，看一下 Memcached 服务器在启动时去哪里查找这个库文件：

```
    [root@nas ~]# LD_DEBUG=libs /usr/local/bin/memcached -v
     31861:find library=libevent-2.0.so.5 [0]; searching
     31861:search cache=/etc/ld.so.cache
     31861:search path=/lib/tls/i686/sse2:/lib/tls/i686:/lib/tls/sse2:/lib/
tls:/lib/i686/sse2:/lib/i686:/lib/sse2:/lib:/usr/lib/tls/i686/sse2:/usr/li
b/tls/i686:/usr/lib/tls/sse2:/usr/lib/tls:/usr/lib/i686/sse2:/usr/lib/i686
:/usr/lib/sse2:/usr/lib (system search path)
     31861:   trying file=/lib/tls/i686/sse2/libevent-2.0.so.5
     31861:   trying file=/lib/tls/i686/libevent-2.0.so.5
     31861:   trying file=/lib/tls/sse2/libevent-2.0.so.5
```

```
31861:   trying file=/lib/tls/libevent-2.0.so.5
31861:   trying file=/lib/i686/sse2/libevent-2.0.so.5
31861:   trying file=/lib/i686/libevent-2.0.so.5
31861:   trying file=/lib/sse2/libevent-2.0.so.5
31861:   trying file=/lib/libevent-2.0.so.5
31861:   trying file=/usr/lib/tls/i686/sse2/libevent-2.0.so.5
31861:   trying file=/usr/lib/tls/i686/libevent-2.0.so.5
31861:   trying file=/usr/lib/tls/sse2/libevent-2.0.so.5
31861:   trying file=/usr/lib/tls/libevent-2.0.so.5
31861:   trying file=/usr/lib/i686/sse2/libevent-2.0.so.5
31861:   trying file=/usr/lib/i686/libevent-2.0.so.5
31861:   trying file=/usr/lib/sse2/libevent-2.0.so.5
31861:   trying file=/usr/lib/libevent-2.0.so.5
31861:/usr/local/bin/memcached: error while loading shared libraries: libevent-2.0.so.5: cannot open shared object file: No such file or directory
```

好了，Memcached 服务器要找 **libevent-2.0.so.5** 的地方找到了，如果没有找到，调试也就停止了。下面我们在系统中查找一下这个库文件在哪里。通过 find 命令查找：

```
[root@nas ~]# find / -name libevent-2.0.so.5
/usr/local/lib/libevent-2.0.so.5
```

找到了，说明我们的系统中存在这个库文件，只是路径不是 Memcached 服务器要找的路径，因此解决如下：

```
[root@nas ~]# ln -s /usr/local/lib/libevent-2.0.so.5 /usr/lib/libevent-2.0.so.5
```

再次运行 Memcached 命令：

```
[root@nas ~]# memcached -h
memcached 1.4.7
-p <num> TCP port number to listen on (default: 11211)
-U <num> UDP port number to listen on (default: 11211, 0 is off)
-s <file> UNIX socket path to listen on (disables network support)
-a <mask> access mask for UNIX socket, in octal (default: 0700)
-l <addr> interface to listen on (default: INADDR_ANY, all addresses)

… //省略
```

3. 安装后的目录结构

```
[root@cache memcached-1.4.5]# tree /usr/local/memcached-1.4.5
/usr/local/memcached-1.4.5
|-- bin
|   '-- memcached
|-- include
|   '-- memcached
|   '-- protocol_binary.h
```

```
    '-- share
    '-- man
    '-- man1
    '-- memcached.1

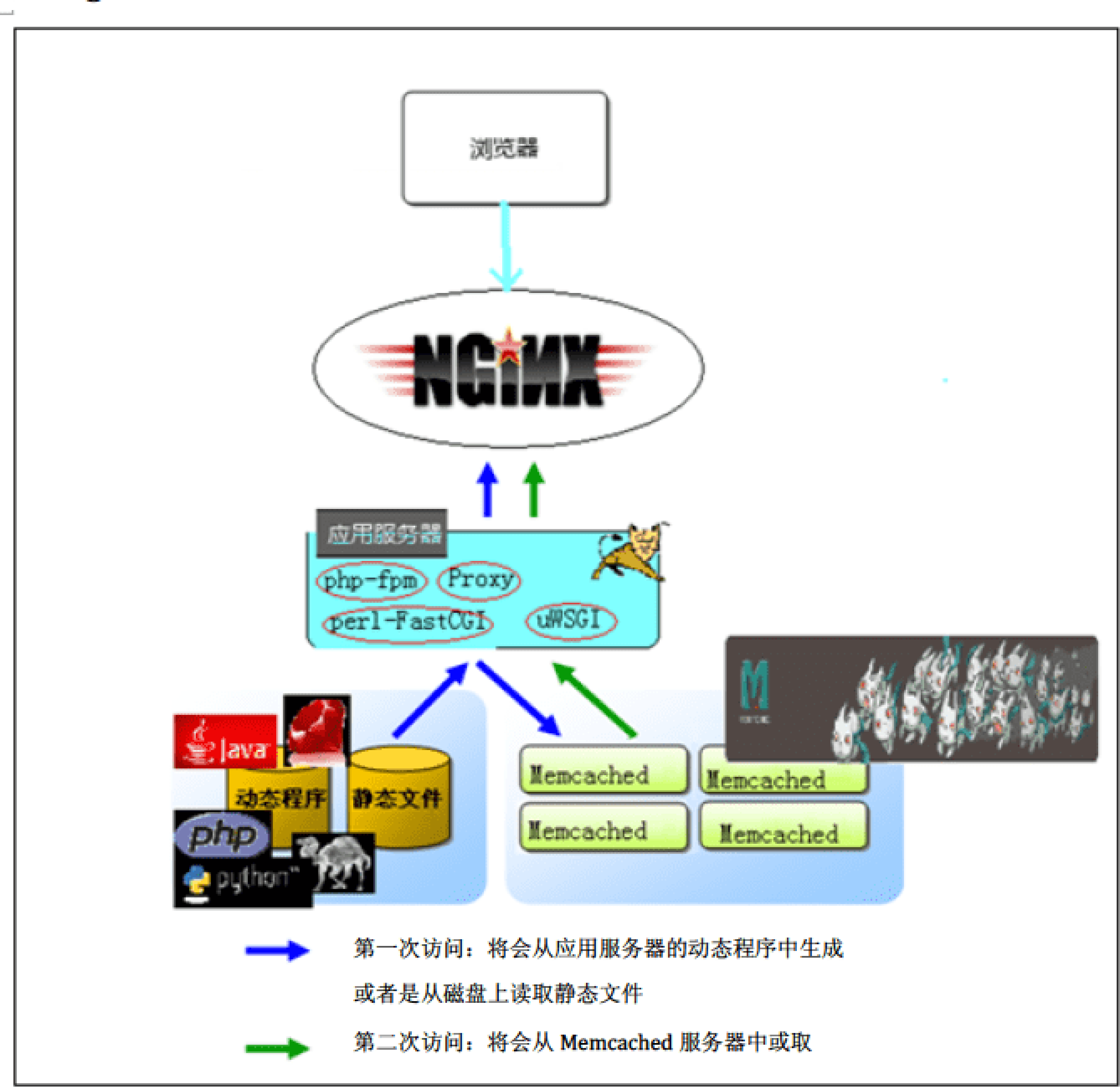
6 directories, 3 files
```

从目录结构可以看出，能够执行的文件只有 bin/memcached，因此只能通过它来启动 Memcached 服务。我们看一下该命令的用法，你可以使用“memcached -h”命令来查看它的用法，例如：

```
[root@cache bin]# ./memcached  -h
memcached 1.4.5
...
```

4. Memcached 的应用图

下图是我们在 Nginx 服务器下的应用。



5. Memcached 命令的相关参数

Memcached 服务器提供了以下命令行参数，因此，在我们启动 Memcached 服务器时可以通过这些参数来做适当（或者叫做按照实际情况）的指定。

参数名称：-p

功能：指定 TCP 协议监听的端口（默认: 11211）。

格式：-p <num>

例如：

```
[root@mail bin]# memcached -d -m 10 -u nobody
[root@mail bin]#
[root@mail bin]# ps -ef|grep memcached
nobody   32525 1  0 17:55 ?00:00:00 memcached -d -m 10 -u nobody
root 32534  9121  0 17:56 pts/900:00:00 grep memcached
[root@mail bin]# lsof -p 32525
COMMAND PID   USER   FD   TYPE DEVICESIZE NODE NAME
memcached 32525 nobody  cwdDIR  253,040962 /
memcached 32525 nobody  rtdDIR  253,040962 /
memcached 32525 nobody  txtREG  253,0  202801 16108004 memcached

… //省略

memcached 32525 nobody   34u  unix 0xd3481d00  5883938 socket
memcached 32525 nobody   35u  unix 0xd3481b00  5883939 socket
memcached 32525 nobody   36u  IPv65883943  TCP *:11211 (LISTEN)
memcached 32525 nobody   37u  IPv45883944  TCP *:11211 (LISTEN)
memcached 32525 nobody   38u  IPv65883948  UDP *:11211
memcached 32525 nobody   39u  IPv45883949  UDP *:11211
```

看黑体字部分，我们没有指定端口号，在这里是 11211，使用的就是默认值，无论是 TCP 的端口号还是 UDP 的端口号都是 11211。

参数名称：-U

功能：指定 UDP 协议监听的端口 （默认: 11211，0 表示关闭）。

格式：-U <num>

例如：

```
[root@mail bin]# memcached -d -m 10 -u nobody -p 11212 -U 11213
[root@mail bin]# ps -ef|grep memcached
nobody   32712 1  0 18:07 ?00:00:00 memcached -d -m 10 -u nobody -p 11212
-U 11213
root 32721  9121  0 18:07 pts/900:00:00 grep memcached
[root@mail bin]# lsof -p 32712
COMMAND PID   USER   FD   TYPE DEVICESIZE NODE NAME
memcached 32712 nobody  cwdDIR  253,040962 /
memcached 32712 nobody  rtdDIR  253,040962 /
memcached 32712 nobody  txtREG  253,0  202801 16108004 /memcached

 … //省略

memcached 32712 nobody   35u  unix 0xdef48b80  5884646 socket
memcached 32712 nobody   36u  IPv65884650  TCP *:11212 (LISTEN)
memcached 32712 nobody   37u  IPv45884651  TCP *:11212 (LISTEN)
```

```
memcached 32712 nobody   38u  IPv65884655  UDP *:11213
memcached 32712 nobody   39u  IPv45884656  UDP *:11213
```

在这里我们看到，TCP 和 UDP 都使用了非默认端口。

参数名称：-s

功能：指定 UNIX 套接字路径，使用 UNIX 套接字来监听。

格式：-s <file>

例如：

```
[root@mail bin]# memcached -d -m 10 -u nobody -s /tmp/memcached.socket
[root@mail bin]# ps -ef|grep memcached
nobody  641  1   0  18:20  ?00:00:00  memcached  -d  -m  10  -u  nobody  -s
/tmp/memcached.socket
root   648  9121  0 18:20 pts/900:00:00 grep memcached
[root@mail bin]# lsof -p 641
COMMAND   PID   USER   FD   TYPE DEVICESIZE NODE NAME
memcached 641 nobody  cwdDIR  253,040962 /
memcached 641 nobody  rtdDIR  253,040962 /
memcached 641 nobody  txtREG  253,0  202801 16108004 memcached

 … //省略

memcached 641 nobody   35u  unix 0xd37e2d80  5885703 socket
memcached 641 nobody   36u  unix 0xd37e2780  5885704 /tmp/memcached.socket
```

参数名称：-a

功能：设置 UNIX 套接字的掩码，格式为八进制（默认：0700）。

格式：-a <mask>

例如：

```
[root@mail bin]# ll /tmp/memcached.socket
srwx------ 1 nobody nobody 0 Sep  9 18:20 /tmp/memcached.socket
```

参数名称：-l

功能：指定监听的网络接口（默认：INADDR_ANY，INADDR_ANY 就是指定为 0.0.0.0 的地址，这个地址事实上表示不确定地址，或“所有地址”、“任意地址”。这里指的是所有 IP 地址）。

格式：-l <ip_addr>

例如：

```
[root@mail  bin]#  /usr/local/memcached-1.4.5/bin/memcached  -d  -m  10  -u
nobody  -l 192.168.3.139
[root@mail bin]# ps -ef|grep memcached
nobody 806 1  0 18:30 ?00:00:00 memcached -d -m 10 -u nobody -l 192.168.4.16
root   813  9121  0 18:30 pts/900:00:00 grep memcached
[root@mail bin]# lsof -p 806
```

```
COMMAND   PID   USER   FD   TYPE DEVICESIZE NODE NAME
memcached 806 nobody cwdDIR  253,040962 /
memcached 806 nobody rtdDIR  253,040962 /
memcached 806 nobody txtREG  253,0  202801 16108004 memcached



memcached 806 nobody  35u  unix 0xde744b805886354 socket
memcached 806 nobody  36u  IPv45886356TCP 192.168.4.16:11211 (LISTEN)
memcached 806 nobody  37u  IPv45886358UDP 192.168.4.16:11211
```

测试一下：

```
[root@mail bin]# telnet 127.0.0.1 11211
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
telnet: Unable to connect to remote host: Connection refused
[root@mail bin]# telnet 192.168.3.139 11211
Trying 192.168.4.16...
Connected to mail.tt.com (192.168.3.139).
Escape character is '^]'.
```

我们可以看到，Memcached 监听的 IP 地址被指定后，就不能够再通过本机的其他 IP 地址接受请求连接了。

参数名称：-d

功能：将 Memcached 服务器作为守护进程运行，即运行在后台。

格式：-d

参数名称：-r

功能：内核文件的最大值限制。

格式：-r

参数名称：-u

功能：使用指定的用户来运行（仅在以 root 用户启动时使用）。

格式：-u <username>

参数名称：-m <num>

功能：指定用于存储缓存条目的最大内存值，单位为 M（默认：64 MB）。

格式：-m <num>

参数名称：-M

功能：在内存耗尽时返回错误（而不是删除缓存条目）。

格式：-M

参数名称：-c

功能：限制同时的最大连接数（默认：1024）。

格式：-c <num>

参数名称：-k

功能：锁定所有分页内存。

> 注意，这里是一个限制值，限制了你可以锁定多少内存，如果分配了超过限定值的操作，将会失败。因此要针对启动 Memcached 的用户进程正确地限制，而不是-u 参数指定的运行用户，在 sh 下，这个操作也可以使用"ulimit -S -l NUM_KB"来完成。对于一个大的缓存来说，这个有点危险，因此查看 README 文档和 Memcached 的主页来查看有关它的配置建议。

格式：-k

参数名称：-v

功能：详细模式（当出现事件循环时，打印 errors/warnings）。

格式：-v

参数名称：-vv

功能：非常详细的模式（这种模式会打印客户端的命令/响应）。

格式：-vv

参数名称：-vvv

功能：极其详细的模式（这种模式会打印内部状态转变）。

格式：-vvv

参数名称：-h

功能：打印帮助并退出。

格式：-h

参数名称：-i

功能：打印 Memcached 和 libevent 许可。

格式：-i

例如：

```
[root@web1 man3]# memcached -i
memcached 1.4.5

Copyright (c) 2003, Danga Interactive, Inc. <http://www.danga.com/>
All rights reserved.

… //省略

This product includes software developed by Niels Provos.

[ libevent ]

Copyright 2000-2003 Niels Provos <provos@citi.umich.edu>
All rights reserved.
```

```
…  //省略
```

看看这些内容还是有必要的。

参数名称：-P

功能：指定保存 PID 文件的路径及文件名，该参数只有在使用-d 选项后才使用。

格式：-P <file>

参数名称：-f

功能：设置块（chunk）大小的增长因子（默认：1.25）。

格式：-f <factor>

参数名称：-n

功能：指定分配给 key+value+flags 的最小空间（默认：48）。

格式：-n <bytes>

参数名称：-L

功能：尽量使用大内存页（如果有效）。增加内存页的大小能够减少 TLB 失误数，并且能提高性能。为了能够从操作系统获取大的内存页，Memcached 将会分配所有的缓存条目到一个大的块（chunk）中。这种机制与所使用的操作系统有关，只有你的操作系统支持才有效。

格式：-L

参数名称：-D

功能：使用<char>作为 key 前缀和 ID 的分隔符，用在状态报告中，默认的<char>是冒号“：”，如果使用该参数指定了分隔符，那么在状态收集中会自动打开，如果没有设置，那么需要通过向服务器发送“stats detail on”命令才会打开。

格式：-D <char>

参数名称：-t

功能：使用的线程数（默认：4）。设置这个值的根据是 CPU 的内核数量，要低于 CPU 的内核数量。

格式：-t <num>

例如：

```
[root@web1 ~]# memcached -d -t 200  -m 300  -u nobody -l 127.0.0.1 -p 11212
-c 25 -P /tmp/memcached1.pid
WARNING: Setting a high number of workerthreads is not recommended.
 Set this value to the number of cores in your machine or less.
```

参数名称：-R

功能：通过该参数来设定一个限制值，定义每一个事件（event）请求的最大值，以便保护处理完成一个单独的客户端请求，一旦一个连接超过了这个值，那么 Memcached 将会试图通过其他的连接来处理 I/O。它的默认值为 20。

格式：-R

参数名称：-C

功能：禁止使用 CAS。

格式：-C

参数名称：-b

功能：设置等待（积压）队列的限制（默认：1024）。

格式：-b

参数名称：-B

功能：设置绑定的协议——可选的有 ascii，binary 或 auto（默认）。

格式：-B

参数名称：-I

功能：设定每一个块（slab）页的大小，默认的大小为 1MB，最小为 1KB，最大为 128MB。调整这个值能够改变对缓存条目大小的限制。当心这个操作同样会增加 slab 的数目（使用-v 来查看），并且会使用 Memcached 的所有内存。

格式：-I

6. 启动 Memcached

下面是启动 Memcached 服务器的两种方式。

第一种方式：

```
[root@cache bin]# ./memcached -p11211 -u nobody -m 64m -vv
```

第二种方式：

```
[root@cache bin]# ./memcached -d -m 10 -u nobody -l 192.168.3.139 -p 11211
-c 25 -P /tmp/memcached.pid  -vv
```

以上是启动 Memcached 的两条命令，其中第二条命令使用了守护进程方式。

启动输出的内容：

```
[root@cache bin]# ./memcached -d -m 10 -u nobody-P /tmp/memcached.pid -vvv
slab class   1: chunk size80 perslab   13107
slab class   2: chunk size   104 perslab   10082
slab class   3: chunk size   136 perslab7710
…
slab class  41: chunk size717184 perslab   1
slab class  42: chunk size   1048576 perslab   1
<36 server listening (auto-negotiate)
<37 send buffer was 109568, now 268435456
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
```

分析如下：

```
  slab class   1: chunk size   80 perslab   13107
```

↑id 号　　↑chunk 的大小↑每个 slab 的 chunk 数量

尽管我们主要讲的是实践，但是有些理论的知识还是要强调一下。这里的 slab 就是相当于 Memcached 提供的缓存，Memcached 服务器对内存的分配单位就是 slab，一个 slab 大小在默认情况下是 1MB，而每一个 slab 又分为若干个 chunk，就像对磁盘或内存的使用一样一层一层地分割直到最小单位，而在这里 chunk 就是最小的单位了，然后在这些 chunk 中保持我们缓存的条目（item），而 chunk 的大小也就完全一样，因此，对于存储一个字节或者是十个字节可能使用的 chunk 数目是一样多的，也就是说也存在着缓存的浪费。

还有一点需要说的是，在每一个 chunk 中除了保存缓存条目的值（value）以外还有结构体、key。如果想要更详细地了解只能去读源代码了，例如，源代码中的语句“-f <factor> chunk size growth factor （default：1.25）\n”，其实这就是命令行中的参数-f：-f <factor> 设置块（chunk）大小的增长因子（默认：1.25），等等。例如，我们把-f <factor>设置为 2，看得会更明白一些：

```
[root@cache bin]# ./memcached -d -m 20-f 2 -c 20 -P /tmp/memcached.pid -vvv
slab class   1: chunk size80 perslab   13107
slab class   2: chunk size   160 perslab6553
slab class   3: chunk size   320 perslab3276
slab class   4: chunk size   640 perslab1638
slab class   5: chunk size  1280 perslab 819
slab class   6: chunk size  2560 perslab 409
slab class   7: chunk size  5120 perslab 204
slab class   8: chunk size 10240 perslab 102
slab class   9: chunk size 20480 perslab  51
slab class  10: chunk size 40960 perslab  25
slab class  11: chunk size 81920 perslab  12
slab class  12: chunk size163840 perslab   6
slab class  13: chunk size327680 perslab   3
slab class  14: chunk size   1048576 perslab   1
<36 server listening (auto-negotiate)
<37 send buffer was 109568, now 268435456
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
<37 server listening (udp)
```

55.4 Memcached 的其他工具

在 Memcached 安装包中有一个 scripts 目录，我们看一下它包括的内容：

```
[root@mail scripts]# tree
.
|-- README.damemtop
|-- damemtop
|-- damemtop.yaml
|-- memcached-init
```

```
    |-- memcached-tool
    |-- memcached.sysv
    '-- start-memcached

    0 directories, 7 files
```

55.4.1 damemtop

我们先来看一下 damemtop 的部分代码：

```
...

use strict;
use warnings FATAL => 'all';

use AnyEvent;
use AnyEvent::Socket;
use AnyEvent::Handle;
use Getopt::Long;
use YAML qw/Dump Load LoadFile/;
use Term::ReadKey qw/ReadMode ReadKey GetTerminalSize/;

...
```

之所以节选这部分代码是想说明该工具使用的 perl 模块，如果以前没有安装过这些模块，那么只有安装之后才可以使用。

1. 安装相关 perl 模块

下面是 AnyEvent、YAML 和 TermReadKey 三个模块的安装。

AnyEvent

```
    [root@mail AnyEvent-5.34] #wget http://cpan.communilink.net/authors/id/
M/ML/MLEHMANN/AnyEvent-5.34.tar.gz
    [root@mail ~]  # tar -zxvf AnyEvent-5.34.tar.gz
    [root@mail ~]  #cd AnyEvent-5.34
    [root@mail AnyEvent-5.34]  # perl Makefile.PL
    [root@mail AnyEvent-5.34]  #make
    [root@mail AnyEvent-5.34]  #make test

    t/00_load...............ok
    t/01_basic..............ok
    t/02 signals............ok
    t/03 child..............ok
    t/04_condvar............ok
```

```
t/05_dns................ok
t/06_socket.............ok
t/07_io.................ok
t/08 idna...............ok
t/handle/01 readline....ok
t/handle/02 write.......ok
t/handle/03 http req....ok
t/handle/04_listen......ok
All tests successful, 1 test skipped.
Files=13, Tests=165,  1 wallclock secs ( 0.85 cusr +  0.13 csys =  0.98 CPU

[root@mail AnyEvent-5.34]  # make install
```

YAML

```
[root@mail ~] # wget http://mirror.osqdu.org/CPAN/authors/id/I/IN
/INGY/YAML-0.73.tar.gz
[root@mail ~]  # tar -zxvf YAML-0.73.tar.gz
[root@mail ~]  #cd YAML-0.73
[root@mail YAML-0.73]  # make
[root@mail YAML-0.73]  # make install
```

TermReadKey

```
[root@mail ~] # wget http://mirrors.ustc.edu.cn/CPAN/authors/id/J/JS
/JSTOWE/TermReadKey-2.30.tar.gz
[root@mail ~]  # tar -zxvf TermReadKey-2.30.tar.gz
[root@mail ~]  #cd TermReadKey-2.30
[root@mail TermReadKey-2.30]  # perl Makefile.PL
[root@mail TermReadKey-2.30]  # make
[root@mail TermReadKey-2.30]  # make install
```

2. 运行 damemtop

我们看一下它的用法：

```
...

sub show_help {
print <<"ENDHELP";
dormando's awesome memcached top utility version v$VERSION

This program is copyright  (c)  2009 Dormando.
Use and distribution licensed under the BSD license.  See
the COPYING file for full text.

contact: dormando\@rydia.net or memcached\@googlegroups.com.
```

```
This early version requires you to edit the ~/.damemtop/damemtop.yaml
(or /etc/damemtop.yaml) file in order to change options.

You may display any column that is in the output of
'stats', 'stats items', or 'stats slabs' from memcached's ASCII protocol.
Start a column with 'all_' (ie; 'all_get_hits') to display the current stat,
otherwise the stat is displayed as an average per second.

Specify a "sort_column" under "top_mode" to sort the output by any column.

Some special "computed" columns exist:
hit_rate (get/miss hit ratio)
fill_rate (% bytes used out of the maximum memory limit)
ENDHELP
exit;
}
```

这是从 damemtop 源码部分节选的，也可以运行“./damemtop --help”命令来查看，是同一个内容。

大致内容说的是，在系统中配置文件的存放位置，根据需要放置，另外还有一些 damemtop 输出的其他指标，我们最关心的指标就是 hit_rate 了，即命中率。

如果在运行 damemtop 时发生以下错误：

```
[root@mail scripts] # ./damemtop
YAML Error: Couldn't open /etc/damemtop.yaml for input:\nBad file
descriptor
   Code: YAML_LOAD_ERR_FILE_INPUT
 at ./damemtop line 543
```

那么肯定是没有找到配置文件，最简单的方法就是将当前目录中的 damemtop.yaml 文件复制到 /etc/damemtop.yaml：

```
[root@mail scripts] # cp damemtop.yaml /etc/
```

分析配置文件 damemtop.yaml

这个文件的格式采用了 YAML（是“YAML Ain't a Markup Language”递归缩写）。下面看一下它的内容：

```
[root@mail scripts] # more /etc/damemtop.yaml
delay: 3
mode: t
top_mode:
sort column: "hostname"
sort order: "asc"
columns:
- hostname
```

```
- all_version
- all_fill_rate
- hit_rate
- evictions
- bytes written
- "2:get hits"
servers:
- 127.0.0.1:11211
- 127.0.0.2:11211
```

- delay：意味着每 3 秒刷新一次。
- mode：表示显示方式，共有三种方式“t”、“？”和“h”，在 damemtop 源代码中是这么定义的：

```
my %display_modes = (
't' => \&display top mode,
'?' => \&display_help_mode,
'h' => \&display_help_mode,
);
```

如果选择使用了 top_mode，那么需要定义以下选项。

- sort_column：定义排序列，如果以“hostname”为排序列，那么可以定义为：

```
sort_column: "hostname"
```

- sort_order：设定排序方式，在这里使用了“asc”方式。
- columns：设定栏目包含的内容，可以称之为包含的条目。在这里选择了 hostname、all_version、all_fill_rate、hit_rate、evictions、bytes_written 和“2:get_hits”。
- Servers：该选项是我们必须配置的，根据我们的 Memcached 服务器安装情况来配置，格式很简单，就是 IP：端口。

根据对配置文件的分析和我们的实际情况，下面对 server 部分进行了修改：

```
servers:
- 127.0.0.1:11211
- 127.0.0.1:11212
```

现在我们再执行./damemtop 命令，将会是类似于以下的输出界面：

```
damemtop: Wed Aug 10 11:26:42 2011 [sort: hostname] [delay: 3s]
hostnameall_version  all_fill_rate  hit_rate  evictions  bytes_written
2:get hits
TOTAL:
NA  NA  NANA NA 9530NA
AVERAGE:
NA  NA  NA100.00%NA 4760NA
127.0.0.1:11211 1.4.56.00814819335937e-06  100.00%0  47400
127.0.0.1:11212 1.4.51.18255615234375e-05  100.00%0  47800
loop took: 0.00349617004394531
```

55.4.2 memcached-init

从文件的名字上就能够看得出 memcached-init 是个 init 脚本文件，我们看一下它的内容：

```
[root@mail scripts]# more memcached-init
#! /bin/sh
#
# skeleton  example file to build /etc/init.d/ scripts.
#   This file should be used to construct scripts for /etc/init.d.
#
#   Written by Miquel van Smoorenburg <miquels@cistron.nl>.
#   Modified for Debian
#   by Ian Murdock <imurdock@gnu.ai.mit.edu>.
#
# Version:  @(#) skeleton  1.9  26-Feb-2001  miquels@cistron.nl
#
### BEGIN INIT INFO
# Provides:  memcached
# Required-Start:$syslog
# Required-Stop: $syslog
# Default-Start: 2 3 4 5
# Default-Stop:  0 1 6
# Short-Description: Start memcached daemon at boot time
# Description:   Enable memcached server
### END INIT INFO


#PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
#DAEMON=/usr/bin/memcached
#DAEMONBOOTSTRAP=/usr/share/memcached/scripts/start-memcached

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
/usr/local/memcached-1.4.5/bin
DAEMON=/usr/local/memcached-1.4.5/bin/memcached
DAEMONBOOTSTRAP=/etc/start-memcached



NAME=memcached
DESC=memcached
PIDFILE=/var/run/$NAME.pid

test -x $DAEMON || exit 0
test -x $DAEMONBOOTSTRAP || exit 0
```

```
set -e

case "$1" in
start)
echo -n "Starting $DESC: "
start-stop-daemon --start --quiet --exec $DAEMONBOOTSTRAP
echo "$NAME."
;;
stop)
echo -n "Stopping $DESC: "
start-stop-daemon --stop --quiet --oknodo --pidfile $PIDFILE --exec $DAEMON
echo "$NAME."
rm -f $PIDFILE
;;

restart|force-reload)
#
#  If the "reload" option is implemented, move the "force-reload"
#  option to the "reload" entry above. If not, "force-reload" is
#  just the same as "restart".
#
echo -n "Restarting $DESC: "
start-stop-daemon --stop --quiet --oknodo --pidfile $PIDFILE
rm -f $PIDFILE
sleep 1
start-stop-daemon --start --quiet --exec $DAEMONBOOTSTRAP
echo "$NAME."
;;
*)
N=/etc/init.d/$NAME
# echo "Usage: $N {start|stop|restart|reload|force-reload}" >&2
echo "Usage: $N {start|stop|restart|force-reload}" >&2
exit 1
;;
esac

exit 0
```

内容不难理解，但需要注意黑体字部分，因为我们的 Memcached 是使用定制安装的，即没有安装在标准的位置，因此，需要对部分内容进行修改。

添加启动：

```
[root@mail scripts]# cp  memcached-init /etc/init.d/memcached
```

```
[root@mail scripts]# chkconfig --add memcached
[root@mail scripts]# chkconfig --list|grep memcached
memcached   0:off  1:off  2:on  3:on  4:on  5:on  6:off
```

到这一步还不行，我们注意到黑体字部分有个 **start-memcached** 文件，因此还要结合该文件。我们接着往下看。

55.4.3 start-memcached

从 memcached-init 文件中看到它需要 start-memcached 文件，因此，有必要看一下该文件：

```
[root@mail scripts]# cat start-memcached
#!/usr/bin/perl -w

# start-memcached
# 2003/2004 - Jay Bonci <jaybonci@debian.org>
# This script handles the parsing of the /etc/memcached.conf file
# and was originally created for the Debian distribution.
# Anyone may use this little script under the same terms as
# memcached itself.

use strict;

if ($> != 0 and $< != 0)
{
print STDERR "Only root wants to run start-memcached.\n";
exit;
}

my $params; my $etchandle; my $etcfile = "/etc/memcached.conf";

# This script assumes that memcached is located at /usr/bin/memcached, and
# that the pidfile is writable at /var/run/memcached.pid

#my $memcached = "/usr/bin/memcached";
my $memcached = "/usr/local/memcached-1.4.5/bin/memcached";
my $pidfile = "/var/run/memcached.pid";


# If we don't get a valid logfile parameter in the /etc/memcached.conf file,
# we'll just throw away all of our in-daemon output. We need to re-tie it so
# that non-bash shells will not hang on logout. Thanks to Michael Renner for the tip
  my $fd reopened = "/dev/null";

sub handle_logfile
```

```
{
my ($logfile) = @_;
$fd_reopened = $logfile;
}

sub reopen logfile
{
my ($logfile) = @_;

open *STDERR, ">>$logfile";
open *STDOUT, ">>$logfile";
open *STDIN, ">>/dev/null";
$fd reopened = $logfile;
}

# This is set up in place here to support other non -[a-z] directives

my $conf_directives = {
"logfile" => \&handle logfile,
};

if (open $etchandle, $etcfile)
{
foreach my $line (<$etchandle>)
{
$line ||= "";
$line =~ s/\#.*//g;
$line =~ s/\s+$//g;
$line =~ s/^\s+//g;
next unless $line;
next if $line =~ /^\-[dh]/;

if ($line =~ /^[^\-]/)
{
my ($directive, $arg) = $line =~ /^ (.*?) \s+ (.*) /;
$conf_directives->{$directive}-> ($arg) ;
next;
}

push @$params, $line;
}

}else{
```

```
$params = [];
}

push @$params, "-u root" unless (grep "-u", @$params) ;
$params = join " ", @$params;

if (-e $pidfile)
{
open PIDHANDLE, "$pidfile";
my $localpid = <PIDHANDLE>;
close PIDHANDLE;

chomp $localpid;
if (-d "/proc/$localpid")
{
print STDERR "memcached is already running.\n";
exit;
}else{
'rm -f $localpid';
}

}

my $pid = fork () ;

if ($pid == 0)
{
reopen_logfile ($fd_reopened) ;
exec "$memcached $params";
exit (0) ;

}else{
if (open PIDHANDLE,">$pidfile")
{
print PIDHANDLE $pid;
close PIDHANDLE;
}else{

print STDERR "Can't write pidfile to $pidfile.\n";
}
}
```

注意黑体字部分，对于“/etc/memcached.conf”文件，我们必须创建它，而 Memcached 的目录也需要修改。

1. 创建 memcached.conf 文件

文件很简单，按照 Memcached 的命令行参数格式书写，每个参数一行：

```
#设定内存使用大小
-m 1024
# 设定监听端口
-p 11211
# 设定运行用户
-u nobody
# 设定监听的 IP 地址
-l 127.0.0.1
```

2. 使用 service 命令启动 Memcached 服务器

如果你现在就是用 service 命令来启动 Memcached 服务器，那么肯定会出现以下提示：

```
[root@mail ~]# service memcached start
Starting memcached: /etc/init.d/memcached: line 45: start-stop-daemon: command not found
```

原因很简单，就是需要 start-stop-daemon。

3. 下载并安装 start-stop-daemon

注意它的安装方式：

```
[root@mail ~]# wget http://developer.axis.com/download/distribution/apps-sys-utils-start-stop-daemon-IR1_9_18-2.tar.gz
[root@mail ~]# tar -zxvf apps-sys-utils-start-stop-daemon-IR1_9_18-2.tar.gz
apps/sys-utils/start-stop-daemon-IR1_9_18-2/
apps/sys-utils/start-stop-daemon-IR1_9_18-2/Makefile
apps/sys-utils/start-stop-daemon-IR1_9_18-2/start-stop-daemon.c
[root@mail ~]#cd apps/sys-utils/start-stop-daemon-IR1_9_18-2/
[root@mail start-stop-daemon-IR1_9_18-2]# gcc start-stop-daemon.c -o start-stop-daemon
[root@mail start-stop-daemon-IR1_9_18-2]# ls
Makefile start-stop-daemon start-stop-daemon.c
[root@mail start-stop-daemon-IR1_9_18-2]# cp start-stop-daemon /bin/
```

生成的 **start-stop-daemon** 就是我们所需要的，根据实际使用情况将其放置在 PATH 中。

4. 再次启动 Memcached 服务器

使用 service 命令启动 Memcached 服务器并查看是否启动：

```
[root@mail ~]# service memcached start
Starting memcached: memcached.
[root@mail ~]# lsof -i:11211
COMMANDPID  USER   FD   TYPE DEVICE SIZE NODE NAME
memcached 1809 nobody  36u  IPv4 144717  TCP localhost.localdomain:11211 (LISTEN)
```

```
memcached 1809 nobody  37u  IPv4 144719  UDP localhost.localdomain:11211
[root@mail ~]# telnet 127.0.0.1  11211
Trying 127.0.0.1...
Connected to localhost.localdomain （127.0.0.1）.
Escape character is '^]'
```

55.4.4 memcached.sysv

如果你饱受了前面那种方法的折磨，那么看看这种方法吧！首先将该脚本文件复制到 init.d 目录，并且执行相关的操作：

```
[root@mail scripts]# cp memcached.sysv  /etc/init.d/memcached
[root@mail scripts]# chkconfig --add memcached
[root@mail scripts]# chkconfig --list|grep memcached
memcached   0:off   1:off   2:off   3:off   4:off   5:off   6:off
[root@mail scripts]# chkconfig --level 35 memcached on
[root@mail scripts]# chkconfig --list|grep memcached
memcached   0:off   1:off   2:off   3:on4:off   5:on6:off
```

下面是该文件的内容（做了相关修改）：

```
[root@mail scripts]# more memcached.sysv
#! /bin/sh
#
# chkconfig: - 55 45
# description:  The memcached daemon is a network memory cache service.
# processname: memcached
# config: /etc/sysconfig/memcached

# Source function library.
. /etc/rc.d/init.d/functions

PORT=11211
USER=nobody
MAXCONN=1024
CACHESIZE=1024
OPTIONS=""

if [ -f /etc/sysconfig/memcached ];then
. /etc/sysconfig/memcached
fi

# Check that networking is up.
if [ "$NETWORKING" = "no" ]
then
```

```
exit 0
fi

RETVAL=0
prog="/usr/local/memcached-1.4.5/bin/memcached"

start () {
echo -n $"Starting $prog: "
# insure that /var/run/memcached has proper permissions
chown $USER /var/run/memcached
daemon /usr/local/memcached-1.4.5/bin/memcached -d -p $PORT -u $USER  -m
$CACHESIZE -c $MAXCONN -P /var/run/memcached/memcached.pid $OPTIONS
RETVAL=$?
echo
[ $RETVAL -eq 0 ] && touch /var/lock/subsys/memcached
}
stop () {
echo -n $"Stopping $prog: "
killproc memcached
RETVAL=$?
echo
if [ $RETVAL -eq 0 ] ; then
rm -f /var/lock/subsys/memcached
rm -f /var/run/memcached.pid
fi
}

restart () {
stop
start
}


# See how we were called.
case "$1" in
start)
start
;;
stop)
stop
;;
status)
status memcached
```

```
;;
restart|reload)
restart
;;
condrestart)
[ -f /var/lock/subsys/memcached ] && restart || :
;;
*)
echo $"Usage: $0 {start|stop|status|restart|reload|condrestart}"
exit 1
esac

exit $?
```

注意黑体字部分，根据实际情况可以进行修改。

使用 service 命令启动 Memcached 服务器并查看启动情况：

```
[root@mail init.d]# service memcached start
Starting /usr/local/memcached-1.4.5/bin/memcached: [  OK  ]
 [root@mail init.d]# lsof -i:11211
COMMANDPID  USER   FD   TYPE DEVICE SIZE NODE NAME
memcached 3835 nobody  36u  IPv6 153648   TCP *:11211 (LISTEN)
memcached 3835 nobody  37u  IPv4 153649   TCP *:11211 (LISTEN)
memcached 3835 nobody  38u  IPv6 153653   UDP *:11211
memcached 3835 nobody  39u  IPv4 153654   UDP *:11211
[root@mail init.d]# telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
```

55.4.5 memcached-tool

先看一下该脚本的内容：

```
[root@mail scripts]# more memcached-tool
#!/usr/bin/perl
#
# memcached-tool:
#   stats/management tool for memcached.
#
# Author:
#   Brad Fitzpatrick <brad@danga.com>
#
# License:
#   public domain.  I give up all rights to this
```

```
#   tool.  modify and copy at will.
#

use strict;
use IO::Socket::INET;

my $host = shift;
my $mode = shift || "display";
my ($from, $to);

if ($mode eq "display") {
undef $mode if @ARGV;
} elsif ($mode eq "move") {
$from = shift;
$to = shift;
undef $mode if $from < 6 || $from > 17;
undef $mode if $to   < 6 || $to   > 17;
print STDERR "ERROR: parameters out of range\n\n" unless $mode;
} elsif ($mode eq 'dump') {
;
} elsif ($mode eq 'stats') {
;
} else {
undef $mode;
}

undef $mode if @ARGV;

die
"Usage: memcached-tool <host[:port]> [mode]\n
   memcached-tool 10.0.0.5:11211 display# shows slabs
   memcached-tool 10.0.0.5:11211# same.  (default is display)
   memcached-tool 10.0.0.5:11211 stats  # shows general stats
   memcached-tool 10.0.0.5:11211 dump   # dumps keys and values
" unless $host && $mode;

$host .= ":11211" unless $host =~ /:\d+/;

my $sock = IO::Socket::INET->new(PeerAddr => $host,
 Proto=> 'tcp');
die "Couldn't connect to $host\n" unless $sock;

if ($mode eq 'dump') {
```

```
my %items;
my $totalitems;

print $sock "stats items\r\n";

while (<$sock>) {
last if /^END/;
if (/^STAT items:(\d*):number (\d*)/) {
$items{$1} = $2;
$totalitems += $2;
}
}
print STDERR "Dumping memcache contents\n";
print STDERR " Number of buckets: " . scalar(keys(%items)) . "\n";
print STDERR " Number of items : $totalitems\n";

foreach my $bucket (sort(keys(%items))) {
print STDERR "Dumping bucket $bucket - " . $items{$bucket} . " total items\n";
print $sock "stats cachedump $bucket $items{$bucket}\r\n";
my %keyexp;
while (<$sock>) {
last if /^END/;
# return format looks like this
# ITEM foo [6 b; 1176415152 s]
if (/^ITEM (\S+) \[.* (\d+) s\]/) {
$keyexp{$1} = $2;
}
}

foreach my $k (keys(%keyexp)) {
print $sock "get $k\r\n";
my $response = <$sock>;
if ($response =~ /VALUE (\S+) (\d+) (\d+)/) {
my $flags = $2;
my $len = $3;
my $val;
read $sock, $val, $len;
print "add $k $flags $keyexp{$k} $len\r\n$val\r\n";
# get the END
$_ = <$sock>;
$_ = <$sock>;
}
}
```

```
}
exit;
}

if ($mode eq 'stats') {
my %items;

print $sock "stats\r\n";

while (<$sock>) {
last if /^END/;
chomp;
if (/^STAT\s+(\S*)\s+(.*)/) {
$items{$1} = $2;
}
}
printf ("#%-17s %5s %11s\n", $host, "Field", "Value");
foreach my $name (sort(keys(%items))) {
printf ("%24s %12s\n", $name, $items{$name});

}
exit;
}

# display mode:

my %items; # class -> { number, age, chunk size, chunks per page,
#total pages, total chunks, used chunks,
#free chunks, free chunks end }

print $sock "stats items\r\n";
while (<$sock>) {
last if /^END/;
if (/^STAT items:(\d+):(\w+)  (\d+)/) {
$items{$1}{$2} = $3;
}
}

print $sock "stats slabs\r\n";
while (<$sock>) {
last if /^END/;
if (/^STAT (\d+):(\w+)  (\d+)/) {
$items{$1}{$2} = $3;
```

```
}
}

print " # Item Size Max age   Pages   Count   Full?  Evicted Evict Time
OOM\n";
foreach my $n (1..40) {
my $it = $items{$n};
next if (0 == $it->{total_pages});
my $size = $it->{chunk_size} < 1024 ?
"$it->{chunk_size}B" :
sprintf("%.1fK", $it->{chunk_size} / 1024.0);
my $full = $it->{free chunks end} == 0 ? "yes" : " no";
printf("%3d %8s %9ds %7d %7d %7s %8d %8d %4d\n",
   $n, $size, $it->{age}, $it->{total pages},
   $it->{number}, $full, $it->{evicted},
   $it->{evicted_time}, $it->{outofmemory});
}
```

注意黑体字部分，“**Usage: memcached-tool <host[:port]> [mode]**”部分为该脚本的使用方法，如果不跟随任何参数来执行该脚本：

```
[root@mail scripts]                    # ./memcached-tool
Usage: memcached-tool <host[:port]> [mode]

   memcached-tool 10.0.0.5:11211 display   # shows slabs
   memcached-tool 10.0.0.5:11211           # same.  (default is display)
   memcached-tool 10.0.0.5:11211 stats     # shows general stats
   memcached-tool 10.0.0.5:11211 dump      # dumps keys and values
```

四种格式

同样是这部分内容，使用方法很简单，该命令之后跟随“IP：端口号”有四种格式。

格式一：./memcached-tool IP：端口号 display

显示 slab：

```
[root@web1 scripts]#  ./memcached-tool 127.0.0.1:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1  80B   1638056s   1   1  no000
  2 104B   1638052s   1   1  no000
  4 176B   1638053s   1   1  no000
  5 224B   1039219s   1   1  no000
  6 280B   1638055s   1   1  no000
  7 352B   1638055s   1   1  no000
  8 440B   1039224s   1   1  no000
  9 552B   1039224s   1   5  no000
 10 696B604236s   1   1  no000
 15 2.1K   1638057s   1   1  no000
```

```
 16 2.6K   1638056s   1   1  no000
 19 5.2K 0s   1   0  no000
 20 6.4K   1638056s   1   1  no000
 21 8.1K   1638055s   1   1  no000
```

格式二：./memcached-tool IP：端口号

这种方式同“格式一”，是一种默认方式，它的默认参数为 display。

格式三：./memcached-tool IP：端口号 stats

显示状态信息：

```
[root@web1 scripts]#  ./memcached-tool 127.0.0.1:11211 stats
#127.0.0.1:11211   Field   Value
 accepting conns   1
   auth cmds   0
 auth errors   0
   bytes   21362
  bytes_read   115529871
   bytes_written   378893173
  cas_badval   0
cas hits   0
  cas misses   0
   cmd flush   0
 cmd get  171875
 cmd_set  106682
 conn_yields   0
   connection_structures   8
curr_connections   5
  curr items  17
   decr hits   0
 decr misses   0
 delete hits   0
   delete_misses   0
   evictions   0
get_hits  110036
  get misses   61839
   incr hits   0
 incr misses   0
  limit maxbytes   314572800
 listen_disabled_num   0
 pid   10369
pointer_size  32
   reclaimed 921
   rusage system8.668682
 rusage user4.146369
 threads   4
```

```
time  1312958794
  total_connections2512
 total_items  106682
  uptime 1639555
 version   1.4.5
```

格式四：./memcached-tool IP：端口号 dump

dump 参数会将内存中现有的缓存对象即“键-值”转存出来。为了简单明了，我们重新启动了一个实例，并且通过 Telnet 的方式存储了一个对象，现在将它转存出来：

```
[root@mail scripts]# ./memcached-tool 127.0.0.1:11211  dump
Dumping memcache contents
  Number of buckets: 1
  Number of items  : 1
Dumping bucket 1 - 1 total items
add logo.jpg 20 1312957668 5
12345
```

可以通过重定向将内容存储到文件中：

```
[root@mail scripts]# ./memcached-tool 127.0.0.1:11211  dump > m_d.txt
Dumping memcache contents
  Number of buckets: 1
  Number of items  : 1
Dumping bucket 1 - 1 total items
[root@mail scripts]# more m_d.txt
add logo.jpg 20 1312957668 5
12345
```

55.5 查看 Memcached 服务的运行情况

telnet 到 Memcached 服务器后有很多的命令可以使用，如 add、get、set、incr、decr、replace、delete 等，除此之外还有一系列的获取服务器信息的命令，这部分命令都是以 stats 开头的。

Memcached 提供了许多命令，可以通过 telnet 工具来执行，当然你也可以使用 PHP 提供的 Memcache::getStats（$cmd）来执行，但我们在这里只使用 telnet 来说明（没办法，我们玩的就是命令行！）

使用 telnet 命令

下面我们使用 telnet 命令来查看一下 Memcached 服务器的工作状态（为了节省版面，我们在下面添加了注解）：

```
[root@cms01 ~]# telnet 192.168.3.139 11211
Trying 192.168.3.139...
Connected to localhost (192.168.3.139).
Escape character is '^]'.
stats                              //使用 stats 命令查看工作状态
```

```
STAT pid 11261                       //Memcached 的进程 pid
STAT uptime 75496                    //服务从启动到现在所经过的时间，单位是秒
STAT time 1293944838                 //服务器所在主机当前系统的时间，单位是秒
STAT version 1.4.5                   //Memcached 服务器的版本
STAT pointer_size 32                 //所在主机操作系统的指针大小，一般为 32 或 64
STAT rusage_user 0.002999            //进程累计使用的用户时间
STAT rusage_system 0.008998          //进程累计使用的系统时间
STAT curr_connections 6              //当前打开的连接数
STAT total_connections 8             //所有的连接数
STAT connection_structures 7         //服务器分配的连接结构数
STAT cmd_get 4                       //执行 get 命令的总数
STAT cmd set 4                       //执行 set 命令的总数
STAT cmd_flush 0                     //执行 flush 命令的总数
STAT get_hits 1                      //get 命令的命中次数
STAT get_misses 3                    // get 命令的失误次数
STAT delete_misses 1                 //delete 命令的失误次数
STAT delete_hits 0                   //delete 命令的命中次数
STAT incr_misses 1                   //incr 命令的失误次数
STAT incr hits 0                     //incr 命令的命中次数
STAT decr_misses 1                   //decr 命令的失误次数
STAT decr_hits 0                     //decr 命令的命中次数
STAT cas_misses 0                    //cas 命令的失误次数
STAT cas_hits 0                      //cas 命令的命中次数
STAT cas_badval 0                    //擦除坏数据值的次数
STAT auth_cmds 0                     //
STAT auth_errors 0                   //
STAT bytes read 477                  //
STAT bytes written 4598              //
STAT limit_maxbytes 10485760         //最大可用内存限制，就是在命令行中-m 的值
STAT accepting_conns 1               //当前接受的连接数
STAT listen_disabled_num 0           //
STAT threads 4                       //最大可使用的线程数
STAT conn_yields 0                   //s
STAT bytes 65                        //所有被存储条目的字节数
STAT curr_items 1                    //当前缓存的条目数
STAT total_items 1                   //缓存条目的总数
STAT evictions 0                     //为了给新的数据项目释放空间，而从缓存移
                                     //除的缓存对象的数目
STAT reclaimed 0                     //
END
```

55.6 服务器的运行情况——详细了解 Memcached 的协议

55.6.1 通信协议

Memcached 的客户端通过 TCP 连接与服务器进行通信，当然对于 UDP 协议也是有效的，请参考下面"UDP 协议"部分。一个运行中的 Memcached 会监听在某个端口，默认的端口为 11211，客户端会连接该端口，并向服务器发送命令，然后读取响应，最后会关闭连接。

要结束一个会话不需要发送任何命令，如果连接不再需要，客户端随时可以关闭连接。需要注意的一点是， Memcached 会鼓励客户端缓存它们的连接，因为这样就不用每一次在客户端需要存储或检索数据时重新打开连接。这种机制是 Memcached 特别设计的，目的是能够在一个特别大的（数以百计，如有必要可能上千）连接下而非常有效地工作。缓存这些连接将会消除与建立 TCP 连接相关的开销。

在 Memcached 协议中，发送的数据有两种类型：文本行和非结构化数据（unstructured data）。文本行被用于命令，包括从客户端发送的命令和从服务器端的响应命令，而非结构化数据则是当客户端向缓存存储或取回检索到的数据时使用，同样的方法，在服务器端，会在接收和发送数据时作为字节流来进行。在非结构化数据方式下，服务器端不关心字节的顺序，也不知道数据是什么内容。对于出现在非结构化数据中的字符没有限制，因此，无法通过特定的字符来"断开"，但是非结构化数据有它自己的方法，那就是通过在文本行中明确指定要传输的字节数，无论是在客户端还是在服务器端。

例如：

```
set abcde 36 0 64
```

这是客户端向服务器发送的命令，最后的一个参数 64，就是传输缓存字节的大小。

在文本行中总是以"\r\n"结束，非结构化数据也是以"\r\n"结束，甚至是"\r"、"\n"或者是任何其他的 8 位字符也可以出现在该数据中。注意，"\r"表示回车符，"\n"表示换行符。因此，当一个客户端从服务器端收取数据的时候，它必须使用数据长度块（这个会被提供）来决定数据块的结束点，而不能依据"\r\n"来决定数据块的结束点，即使是正好使用"\r\n"来表示，也是不行的。

55.6.2 键（Key）

数据借助一个 Key 存储在 Memcached 中，Key 是一个文本字符串，它唯一地标识了存储在 Memcached 中的缓存条目，用于客户端存储或者是重复使用感兴趣的数据。对于一个 Key 的长度，当前的限制是 250 个字节（当然，一般的客户端使用不了这么长的 Key），Key 的名称中禁止包含控制字符或空格字符。

55.6.3 命令

Memcached 的命令有三种类型。

存储数据命令（有六个命令：set、add、replace、append、prepend 和 cas）请求服务器通过一个 Key 来存储一些数据。客户端发送一个命令行，然后是一个数据块（我的理解是要获取的数

据的 Key），最后客户端会期待着服务器返回的响应行，它会指明期待的操作是成功还是失败。

取回数据命令（有两个命令：get 和 gets）请求服务器取回数据（重新从 Memcached 中获取存储的数据），相当于一组 Key（在一个请求中获取一个或多个 Key）。

客户端通过命令行发送一个获取数据的命令，该命令行包含了所有的请求 Key，之后，服务器查找到的每一个条目都会发送到客户端，每一个发送的条目都会有相应的响应信息，然后就是数据块，一个数据块表示一个条目数据，直到服务器发送完成并且以“END”字符串结束响应行。

所有其他的命令不涉及非结构化数据，在它们中，客户端发送一个命令行，然后期待（依赖于命令）一行或者是多行的响应，最后响应会以“END”结束。

一个命令行总是由一个命令名开始，然后会跟一个参数（如果它有参数），命令和参数之间以空格分隔，命令对大小写敏感，且只能用小写字母。

55.6.4 过期时间

有些命令会涉及客户向服务器发送一些时限，这些设置会关系到一个条目或是由客户端请求的操作。在所有这些情况下，实际被发送的时间值可能是 UNIX 时间，它是一个 32 的值，是从 1970 年 1 月 1 日开始的时间值，或者可能是一个从当前时间开始的值。对于后一种情况，指定的数值不能超过 60*60*24*30（这是 30 天的一个秒表示形式），如果客户端发送的这个值大于该值，那么服务器会将客户端发送的值看做是一个 UNIX 时间，而不是从当前时间开始的相对值。

55.6.5 错误字符串

每一个从客户端发送的命令可能被服务器的一个错误字符串所回答，这些错误的字符串来自以下三种类型。

第一种类型：

```
"ERROR\r\n"
```

它表示客户端发送了不存在的命令。

第二种类型：

```
"CLIENT_ERROR <error>\r\n"
```

它表示某些客户端在输入行中的错误，例如 “CLIENT_ERROR bad command line format”，客户端在命令行中在某种程度上没有遵照本协议。<error>是一个可读懂的错误字符串，例如，上面的错误表示“客户端命令行格式错误”。

第三种类型：

```
"SERVER_ERROR <error>\r\n"
```

它表示服务器端错误，Memcached 服务器端阻止了命令的执行。<error>是一个可读懂的字符串，在服务器发生严重错误的情况下，这种情况一般不会发生，Memcached 服务器会在发送完成该错误消息后关闭连接，这是唯一的一种服务器关闭客户端连接的情况。

以下是对各个命令的描述，这些错误行不会再被具体提到，但是客户端必须允许它们存在的可能性。

55.6.6 存储数据的命令

首先，客户端发送一个类似于下面的命令行：

```
<command name> <key> <flags> <exiptme> <bytes> [noreply]\r\n
```

或：

```
cas <key> <flags> <exptime> <bytes> <cas unique> [noreply]\r\n
```

- <command name> 可以是“set”，“add”，“replace”，“append”或“prepend”。
  - set：它的功能是存储指定的数据。
  - add：它的功能是存储指定的数据，但是仅在 Memcached 服务器没有持有该 Key 的情况下才会存储该数据。
  - replace：它的功能是存储指定的数据，但是仅在 Memcached 服务器持有该 Key 的情况下才存储该数据。
  - append：它的功能是存储指定的数据，但是它会存放在已存在数据的尾部，换句话说就是紧跟着已存在的数据存储。
  - prepend：意思是存储指定的数据，但是它会存储在已存在的数据之前，与 append 命令正好相反。
  - 命令 append 和 prepend 不接受 flags 和 exptime，它们只是更新已存在的数据部分，而忽略新的 flags 和 exptime 的设置。
- cas：是一个检查并设置操作，检查在前，设置在后，它的意思是存储指定的数据，但是仅在至从（我）上次取过之后就没有被更新过。
  - <key>：是一个键，客户端就是通过它来查询存储的数据。
  - <flags>：是一个任意的 16 位无符号整数（但是通过十进制写出），Memcached 服务器除了存储数据外还存储了该值，当一个条目被找到（就是在 Memcached 的缓存中被检索到）会将该值返回。客户端可以使用这个值作为一个位字段（bit field）来存储特殊的数据信息，该位字段对服务器不透明。注意，在 Memcached 1.2.1 或更高的版本中，“flags”的值可能是 32 位，而不是 16 位，但是为了兼容老的版本，你可能想限制你自己而使用 16 位。
  - <exptime>：设置缓存条目的生存期，如果设置为 0，那么表示该条目的生存期为无限长，换句话说就是没有消亡期，为了给其他的条目腾挪地方，它还有可能被从缓存中删除。如果设置为非 0 数字，使用 UNIX 时间或是从当前时间开始的偏移值，单位为秒，当到达消亡的期限时，保证客户端不会从 Memcached 的缓存中找到，需要注意的一点是，不是以客户端的时钟为准，而是以服务器的时间度量。
  - <bytes>：设定跟随的字节的长度，需要注意的一点是，不包括定界符“\r\n”。另外 <bytes>可以为 0，跟随 Key 的只是一个空的数据块。
  - <cas unique>：是现有条目的一个唯一的 64 位值，是 Memcached 服务器端为缓存的条目分配的一个 64 位唯一序列号，实际就是一个计数器，这个序号会随着缓存条目的增多而增大，如果将其中的缓存条目删除了，但计数器的递增不会改变，计数器的最大值就是 Memcached 从启动以来缓存过条目的最大数，当然包括被删

除的缓存条目。当使用“cas”升级数据时，客户端应该使用这个值，该值由“gets”命令返回。

- <noreply>：该选项指明了服务器不发送答复。注意，如果请求行畸形难看，那么 Memcached 将不能够有效地解析“noreply”选项。在这种情况下，它会向客户端发送错误报告，不再从客户端读取数据，并且会中断该行为，因此，客户端应该只建立有效的请求。

命令行之后就是客户端发送的数据块：

```
<data block>\r\n
```

<data block> 是任意长度的 8 位（8-bit）数据，它的长度由上面命令行中指定的参数<bytes>决定，数据的长度不能大于这个值也不能小于这个值。

当发送完命令行和数据块之后，客户端会期待一个类似于以下格式的回复：

- “STORED\r\n”：表示成功发送 。
- “NOT_STORED\r\n”：指示出数据没有被存储，但是不是因为错误。这种情况通常出现在使用命令“add”或“replace”的时候，由于它们指定的条件没有找到，或者是相应的条目被删除队列。参考“delete“命令。
- “EXISTS\r\n”：指示出你试图通过“cas”命令来修改存储的条目至从你上次取过后就被更新过了。
- “NOT_FOUND\r\n”：指示出你试图通过“cas”命令来修改存储的条目不存在或者是已经被删除了。

举例

set 命令添加数据：

```
set a  36 0 5         //第一次 set
12345
STORED
get a                 //第一次查看 a 的值
VALUE a 36 5
12345
END
set a  36 0 6         //第二次 set
123456
STORED
get a                 //第二次查看 a 的值
VALUE a 36 6
123456
END
```

add 命令添加数据：

```
add b  36 0 5        //第一次 add
abcde
STORED
get b                //第一次查看 b 的值
```

```
VALUE b 36 5
abcde
END
add b  36 0 6          //第二次 add
abcdef
NOT STORED
get b                  //第二次查看 b 的值
VALUE b 36 5
abcde
END
```

replace 命令替换数据：

```
get a                  //首先查看 a 的值
VALUE a 36 6
123456
END
replace a 36 0 6       //替换 a 的值
654321
STORED
get a                  //再次查看 a 的值
VALUE a 36 6
654321
END
```

append 命令在原有数据的最后追加数据：

```
get a                  //首先查看 a 的值
VALUE a 36 6
654321
END
append a 36 0 6        //追加数据
adcdef
STORED
get a                  //再次查看 a 的值
VALUE a 36 12
654321adcdef
END
```

prepend 命令在原有数据的最前面添加数据：

```
get a
VALUE a 36 12
654321adcdef
END
prepend a 36 0 8
Good!!!
```

```
STORED
get a
VALUE a 36 20
Good!!! 654321adcdef
END
```

cas 检查并设置数据：

```
 add c 20 0 5         //添加 c
12345
STORED
gets c                //查看 c
VALUE c 20 5 10
12345
END
cas c 20 0 5 10       //第一次 cas
abcde
STORED
cas c 20 0 5 10      //第二次 cas，可见是不能修改的，
12345                //我们使用 gets 命令再次查看时
EXISTS              //将会发现 c 的<cas unique>部分已经
gets c              //发生了改变，由前面的 10 变为 11，这意味着缓存的条目已被重新存储
VALUE c 20 5 11
abcde
```

55.6.7　获取数据的命令

获取数据的命令“get”和“gets”操作类似，如下：

```
get <key>*\r\n
```

或

```
gets <key>*\r\n
```

<key>*　表示一个或多个 Key 字符串，这些字符串之间由空格字符分隔。执行此命令之后，客户端将会期待得到 0 个或多个条目的返回，每一个被接收到的条目作为一个文本行并跟随一个数据块，当所有的条目被传送完毕后，服务器将会发送字符串“END\r\n”来指示响应结束。由服务器发送回来的每一个条目类似如下格式：

```
VALUE <key> <flags> <bytes> [<cas unique>]\r\n
<data block>\r\n
```

- <key>：被发送的条目的 Key。
- <flags>：通过存储命令设置的标志值。
- <bytes>：后面跟随的数据块的长度。注意，不包括定界符“\r\n”。
- <cas unique>：是一个 64 为整数，它唯一地标识了一个确切的条目。换句话说就是 Memcached 缓存系统中唯一的序列号。
- <data block>：该条目所代表的数据。

如果一些 Key 出现在取回的请求中，但是没有被 Memcached 服务器发送回，这意味着该服务器没有持有该条目，原因是它们从来就没有被缓存过，或者是曾经缓存过，但是为了腾出空间而被删除了，或者是过生存期了，也有可能是被客户端明确删除了。

有关这两个命令的例子可以参考前面讲的“存储数据”部分，在这里再举一个例子对比一下这两个命令的区别：

```
get c   //使用 get 命令获取数据
VALUE c 20 5
12345
END
gets c  //使用 gets 命令获取数据
VALUE c 20 5 12
12345
END
```

在这个例子中，我们分别使用了 get 和 gets 两个命令，得到的结果唯一的区别就是使用 gets 命令比 get 命令多一个值，而这个值就是在现有缓存中能够唯一标识所有缓存条目的 64 位唯一值。

55.6.8 删除数据的命令

命令“delete”允许明确地删除一个条目：

```
delete <key> [<time>] [noreply]\r\n
```

- <Key>：被删除条目的 Key。
- <time>：它表示的是一个以秒为单位的时间或是直到某一时刻的一个 UNIX 系统时间，一旦执行了带有该选项的 delete 命令，指定的 Key 就会被移动到 delete 队列，在这个指定的时间段内服务器会拒绝对该 Key 进行 add 和 replace 命令，也无法再使用获取数据的命令来获取它的内容，直到这个“生存期”到期，它的内容就会被从缓存中彻底删除。该选项为可选项，默认值为 0，表示被立即清除，并且随后它的 Key 也就会被其他的存储命令成功使用。
- <noreply>：该选项也是一个可选参数，指示服务器不发送答复，参考前面的“存储数据的命令”部分。

对于该命令以下是可能的响应：

- DELETED\r\n：表示缓存条目被成功删除。
- NOT_FOUND\r\n：表示指定的条目没有被找到。

另外还有一个 flush_all 命令：

查看下面的 flush_all 命令，该命令用于让所有存在的缓存条目立即失效。

举例

只跟有 Key 的 delete 命令：

```
set d 36 0 5            //设置 d
good!
STORED
get d                   //查看 d
```

```
VALUE d 36 5
delete d              //删除 d
DELETED
get d                  //查看 d，此时已无数据
END
```

带有时间值的 delete 命令：

```
delete c 1296262920  //通过 UNIX 时间指定删除时间
CLIENT_ERROR bad command line format.  Usage: delete <key> [noreply]
delete c 3600         //通过秒数指定删除时间
CLIENT_ERROR bad command line format.  Usage: delete <key> [noreply]
delete c 3600 noreply
```

这两种指定时间的删除方式得到的是同样的结果，看它的提示是说该指令的用法为：delete <key> [noreply]，但是官方的协议中讲述（参考上面的内容）是可以添加时间值，可能是我使用的这个版本不支持吧！

55.6.9 增加/减少数据的命令

命令 incr 和 decr 被用于改变某些条目中适当位置的数据内容，对它进行增加或减少的操作，用于条目的数据被看做是十进制表示法的 64 位无符号整数，如果当前的数据值不符合这样的表示法，那么该命令会将它的值作为 0 处理。同样，被修改的条目必须已经存在，那么命令 incr 或 decr 才会正常工作，在这些命令修改一个不存在的条目时，Memcached 服务器不会将值看做 0 来处理，相反，而是当做操作失败。

客户端发送的命令行：

incr 增加指定的值：

```
incr <key> <value> [noreply]\r\n
```

或

decr 减少指定的值：

```
decr <key> <value> [noreply]\r\n
```

- <key>：这里的 key 是客户端希望修改的条目。
- <value>：客户端想要对一个缓存条目增加/减少的大小，它是一个 64 位的无符号整数，用十进制表示。
- <noreply>： 这是一个可选参数，它通知服务器不发送应答。具体可参考前面的“存储数据的命令”部分。

下面是可能出现的响应：

- NOT_FOUND\r\n：表示指定条目的值没有找到；
- <value>\r\n：这里的<value> 是一个新的条目数值，是在执行增加/减少操作后的值。

使用 incr 和 decr 这两个命令时需要注意：

- 命令 decr 会产生下溢（underflow）：如果一个客户端试图将存储在缓存的值减少到“0”以下，那么新的值将会是“0”，而不会产生负数；
- 命令 incr 会产生溢出（overflow）：如果客户端提交的值和存储在缓存中的值的和大于

64 位无符号数的最大值，那么将会产生溢出，溢出后将会成为是 64 位的掩码，参见下面的例子。

举例

通过命令 incr 来增加数据值，存储的数据为数值：

```
add jh123 36 0 8
11111111
STORED
get jh123
VALUE jh123 36 8
11111111
END
incr jh123 77777777
88888888
get jh123
VALUE jh123 36 8
88888888
END
```

数据的值为字符串：

```
add jh124 36 0 8
abcdefgh
STORED
incr jh124 22222222
CLIENT_ERROR cannot increment or decrement non-numeric value
```

最后的一行足以说明问题：不能增加或减少一个非数字的值。

通过命令 decr 来减少数据值，存储的数据为数值：

```
get jh123
VALUE jh123 36 8
88888888
END
decr jh123 10000000       //第一次减去一个小于存储在缓存中的数
78888888
decr jh123 8888          //同样减去一个小于存储在缓存中的数
78880000
decr jh123 100000000     //第三次减去一个大于存储在缓存中的数
0                        //我们看结果变为 0
get jh123
VALUE jh123 36 8
0
END
```

溢出：

```
incr jh123 9999999999999999999
17767279631452241928
```

```
incr jh123 9999999999999999999
9320535557742690311
incr jh123 9999999999999999999
873791484033138694
incr jh123 9999999999999999999
10873791484033138693
incr jh123 9999999999999999999
2427047410323587076
incr jh123 9999999999999999999
12427047410323587075
```

注意黑体字的变化。

55.6.10 查询存储状态的命令

命令 stats 被用于查询有关 Memcached 服务器的状态和其他的内部数据。它有两种格式，没有参数格式：

```
stats\r\n
```

这将会使得服务器输出多方面的状态信息和设定，参考下面的相关文档。

另外一个格式就是带有参数的格式：

```
stats <args>\r\n
```

依赖于指定的<args>，各种内部数据将会由 Memcached 服务器发送至客户端，对于发回的各种参数和数据在这个版本的协议中没有相关的文档说明，主要是为了开发的方便（就是更改的方便！）。

55.6.11 多方面统计命令

在上述中，如果 Memcached 服务器收到一个没有参数的 stats 命令，例如：

```
stats
STAT pid 28421
STAT uptime 21663
STAT time 1295943998
STAT version 1.4.5
STAT pointer_size 32
STAT rusage_user 0.033994
STAT rusage_system 0.048992
STAT curr_connections 10
STAT total connections 14
STAT connection structures 11
STAT cmd get 54
STAT cmd set 27
STAT cmd_flush 0
STAT get_hits 48
STAT get_misses 6
```

```
STAT delete_misses 0
STAT delete_hits 1
STAT incr_misses 0
STAT incr_hits 22
STAT decr_misses 0
STAT decr_hits 3
STAT cas_misses 0
STAT cas_hits 3
STAT cas_badval 5
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 3489
STAT bytes_written 4168
STAT limit_maxbytes 20971520
STAT accepting_conns 1
STAT listen_disabled_num 0
STAT threads 4
STAT conn_yields 0
STAT bytes 556
STAT curr_items 8
STAT total_items 18
STAT evictions 0
STAT reclaimed 0
END
```

归纳一个上面的返回行，服务器端会发回许多类似于下面的行：

```
STAT <name> <value>\r\n
```

服务器端以发送下面的行来结束这个列表：

```
END\r\n
```

在每一个统计行，<name>是一个名字，它代表了该状态的功能，<value>是一个值，它是<name> 的数据，表明了具体的值。在下面的列表中，是由发送“stats”命令而将会得到的结果，并且在列表中指明了它的类型并解释了它的功能。在类型一列中，“32u”表示使用 32 位无符号整数，而“64u”则代表了 64 位无符号整数，“32u:32u”意味着是有冒号（:）隔开的两个无符号整数。

- pid 32u：Memcached 的进程 pid。
- uptime 32u：服务从启动到现在所经过的时间，单位是秒。
- time 32u：服务器所在主机当前系统的时间，单位是秒。
- version string：Memcached 服务器的版本。
- pointer_size 32 ：所在主机操作系统的指针大小，一般为 32 或 64。
- rusage_user 32u:32u：进程累计使用的用户时间（单位：微秒）。
- rusage_system 32u:32u：进程累计使用的系统时间（单位：微秒）。
- curr_items 32u：当前缓存的条目数。

- total_items 32u：从 Memcached 服务器启动到现在缓存条目的总数。
- bytes 64u：所有被存储条目的字节数。
- curr_connections 32u：当前打开的连接数。
- total_connections 32u：从 Memcached 服务器运行到现在所有的连接数。
- connection_structures 32u：服务分配的连接结构数。
- cmd_get 64u：执行 get 命令的总数，也就是取回已缓存条目的次数。
- cmd_set 64u：执行 get 命令的总数，也就是执行存储请求的次数。
- get_hits 64u：get 命令的命中次数。
- get_misses 64u：get 命令的失误次数。
- evictions 64u：为了给新的数据项目释放空间，而从缓存移除缓存对象的数目。
- bytes_read 64u：Memcached 服务从网络读取的字节数。
- bytes_written 64u：由该 Memcached 服务向网络发送的字节数。
- limit_maxbytes 32u：允许该 Memcached 服务器最大可用内存限制，就是在命令行中-m 的值。
- threads 32u：最大可使用的线程数（参考 doc/threads.txt 文件）。

55.6.12 条目统计命令

官方文档说得很清楚，下面的描述在将来的版本中有可能会被更改，因此不要把它作为依据，而是作为一种方法去认识 Memcached 而已。

stats 命令的参数为“items”，它返回了每一个有关存储在 slabclass 的（特定大小的 chunk 的组）“items”的信息。返回的数据格式如下：

```
STAT items:<slabclass>:<stat> <value>\r\n
```

服务器将会以下面的输出行结束该列表：

```
END\r\n
```

1. stats slabs 和 stats items 命令

这两个命令描述了内存的大小和使用情况，通过 stats slabs 命令按照 slabclass 分类 ID 显示出条目使用 slab 情况的统计信息：

```
stats slabs            //执行 stats slabs 命令
STAT 1:chunk_size 80
STAT 1:chunks per page 13107
STAT 1:total pages 1
STAT 1:total_chunks 13107
STAT 1:used_chunks 2
STAT 1:free_chunks 0
STAT 1:free chunks end 13105
STAT 1:mem requested 115
STAT 1:get hits 2
STAT 1:cmd_set 2
```

```
STAT 1:delete_hits 0
STAT 1:incr_hits 0
STAT 1:decr_hits 0
STAT 1:cas_hits 0
STAT 1:cas_badval 0
STAT 12:chunk_size 1096
STAT 12:chunks_per_page 956
… //省略
STAT 12:cas_badval 0
STAT 13:chunk_size 1376
STAT 13:chunks_per_page 762
… //省略
STAT 13:cas_badval 0
STAT 16:chunk_size 2696
… //省略
STAT 16:cas_badval 0
STAT active_slabs 4
STAT total_malloced 4190896
END
```

使用 stats items 命令会显示更详细的信息：

```
stats items                    //执行 stats items 命令
STAT items:1:number 2
STAT items:1:age 33
STAT items:1:evicted 0
STAT items:1:evicted_nonzero 0
STAT items:1:evicted_time 0
STAT items:1:outofmemory 0
STAT items:1:tailrepairs 0
STAT items:1:reclaimed 0
STAT items:13:number 1
STAT items:13:age 2578
STAT items:13:evicted 0
STAT items:13:evicted_nonzero 0
STAT items:13:evicted_time 0
STAT items:13:outofmemory 0
STAT items:13:tailrepairs 0
STAT items:13:reclaimed 0
STAT items:16:number 1
… //省略
END
```

- number：目前存储在这个类型（class）下的条目，对于过期的条目不会自动排除在外。
- age：在算法 LRU 下，最老的条目经历的时间。
- evicted：在生存期满之前，在算法 LRU 中一个条目不得不被驱赶出缓存的次数。

- outofmemory：该 slabclass 不能够存储新条目的次数。这种情况一般出现在启动 Memcached 服务器时使用了-M 参数，或是在清除条目时失败所致。

注意，这里显示的是关于存在的 slabs 信息，因此空的缓存将会返回空集，例如：

```
stats items   //执行 stats items 命令
END
stats slabs   //执行 stats slabs 命令
STAT active_slabs 0
STAT total malloced 0
END
```

附加说明：本节描述的状态信息在将来的版本中有可能会改变。

2. stats sizes 命令

命令 stats 再加一个参数“sizes”，将会返回存储在缓存中的所有大小的缓存条目及其数目。

警告：这个命令将会锁定缓存！它将遍历每一个条目，并且检测它的大小。然而由于这个操作非常快，如果你有许多缓存条目，那么你的这个操作将会阻止 Memcached 服务在几秒钟内不会提供服务。

例如：

```
stats sizes  //执行 stats sizes 命令
STAT 64 3
STAT 1184 1
STAT 2272 1
END
```

该命令发回的数据格式如下：

```
<size>   <count>\r\n
```

↑条目的大小↑条目的数量

注意一点：这个条目的大小是由三部分组成的——32 位的哈希值、key 的长度和 value 的长度

- size：这里的 size 是一个近似值，它是 32 的倍数；
- count：该“size”下存在的条目数。

服务器会使用下面的行结束这个列表：

```
END\r\n
```

我们再举一个例子：

```
set abcde 36 0 64         //存储一个 key 为 abcde，大小为 64 位的条目
1234567890123456789012345678901234567890123456789012345678901234
STORED
stats sizes               //执行 stats sizes 命令
STAT 64 3
STAT 128 1                //而得到的条目却被列为 128 的 size 中
STAT 1184 1
STAT 2272 1
```

```
END
```

在这个例子中，我们从根本上看到了缓存中所有的条目。如果你缓存的条目足够多，并且足够丰富（所谓的丰富就是每一个 slab 级别都有存储），那么你会看到每一个 STAT 之间的差是 32。

你可以根据这个提示来决定是否调整 slab 增长因子，这样可以节省内存的开销，或者说减少内存空间的浪费。例如：如果你存储的条目中大多数条目小于 200 字节，那么产生更多的小范围的类别以便适合更多的条目存储在这些 slab 类别中。

3. slab 统计

附加说明：本节描述的状态信息在将来的版本中有可能会改变。

命令 stats 使用参数“slabs”将会返回每一个 slabs 在 Memcached 运行期间的相关信息，除了一些总计外，这将会包含每一个 slab 的信息。

```
stats slabs //执行 stats slabs 命令
STAT 1:chunk_size 80
STAT 1:chunks_per_page 13107
STAT 1:total_pages 1
STAT 1:total_chunks 13107
STAT 1:used_chunks 2
STAT 1:free_chunks 0
STAT 1:free_chunks_end 13105
STAT 1:mem_requested 115
STAT 1:get_hits 0
STAT 1:cmd_set 2
STAT 1:delete_hits 0
STAT 1:incr_hits 0
STAT 1:decr_hits 0
STAT 1:cas_hits 0
STAT 1:cas_badval 0
… //省略部分
STAT active_slabs 2
STAT total_malloced 2096992
END
```

返回的数据格式如下：

```
STAT <slabclass>:<stat> <value>\r\n
STAT <stat> <value>\r\n
```

服务器会使用下面的行结束这个列表：

```
END\r\n
```

项目名称和功能如下。

- chunk_size：每个 chunk 使用的空间数量。对于条目的存储，它会被存储到一个与它近似大小的 chunk 中。

- chunks_per_page：一个 page 中有多少个 chunk，这里的 page 就是 slab，一个 slab 默认是 1M，Memcached 会将 page 分配给 slab，一个 slab 一个 page，然后再将 slab 划分为 chunk。换句话说就是每一个 slab 就是一个 page。
- total_pages：该 slab 类分配到的 page 数量。
- total_chunks：该 slabclass 拥有的 chunk 数量。
- used_chunks：被使用的 chunk 数量，就是已被缓存条目占用 chunk。
- free_chunks：未被使用的 chunk 数量，就是没有被缓存条目使用的 chunk 或者是通过删除条目而又释放的 chunk。
- free_chunks_end：在一个 slab 中的最后一页的空闲 chunk 数量。
- active_slabs：已被分配出的 slab 类。
- total_malloced：缓存中总共的 slab 页数量。

55.6.13 其他命令

1. flush_all 命令

flush_all 命令有一个可选的数字参数，它的执行总会成功，服务器端返回的值总是“OK\r\n”，除非在命令行中使用了“noreply“。flush_all 的功能是立即使所有的缓存条目无效，这是默认的操作。当然也可以在该命令的后面添加一个生存期，直到期满时再使得所有的缓存条目无效。当缓存的条目失效后，对于每一个想重新获取条目的命令都没有相应的返回值，总是以唯一的“END”结束，除非是在使用了 flush_all 命令之后又以同样的 key 存储了缓存条目。实际上使用 flush_all 命令后并没有释放被已存在条目占用的内存，下面来看一下：

```
flush_all
OK
get a
END
stats items
STAT items:1:number 6
STAT items:1:age 16876
STAT items:1:evicted 0
STAT items:1:evicted_nonzero 0
STAT items:1:evicted_time 0
STAT items:1:outofmemory 0
STAT items:1:tailrepairs 0
STAT items:1:reclaimed 0
STAT items:2:number 1
STAT items:2:age 241
STAT items:2:evicted 0
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 0
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
```

```
STAT items:2:reclaimed 0
END
get a
END
```

首先执行 flush_all 命令，服务器返回的一定是 OK，然后再去获取一个条目 a，返回的结果是 END，但是被存储的条目并没有被真正地删除，而只是访问不到其内容而已，它们所占用的缓存空间终将会被新的缓存条目所覆盖。

看下面的一个例子：

```
add jkl 36 0 3          //添加缓存条目 jkl
123
STORED
flush_all 120           //120 秒后使得所有缓存条目失效
OK
get jkl                 //立即查看缓存条目 jkl
VALUE jkl 36 3
123
END
append jkl 36 0 3       //修改缓存条目 jkl
321
STORED
get jkl                 //立即查看修改后的缓存条目 jkl
VALUE jkl 36 6
123321
END
get jkl                 //120 秒后再次查看缓存条目 jkl
END
```

可以看到当执行完“flush_all 120”命令后，在这个 120 秒之内的时间段内，还可以对缓存条目进行操作，一旦超过这个期限，那么客户端所有的操作将会无效。例如，当你缓存的网页不再使用（换句话说就是，领导说了要撤稿，你可以在源网站上将网页删除，但是客户访问的却是缓存，这时“flush_all”命令就有用了），那么你可以通过该命令将缓存中的现有条目废除。

使用带有延时参数的 flush_all 命令的目的在于：假如你有一个 Memcached 服务器池（就是一组功能相同的服务器），当你想清除所有服务器上的所有缓存条目时，如果你在所有的服务器上同时执行一个不带延时参数的 flush_all 命令，那么会出现所有的 Memcached 服务器需要重新生成缓存的一个过程，如果这个过程同时发生，那后台数据库或者 Web 网站将会受到非常大的压力，巨大的访问量会将网站拖垮直到不能访问为止。

因此，你需要一个带有延时参数的 flush_all 命令，假如你有 20 台 Memcached 服务器，那么第一台可以执行 flush_all，而第二台执行 flush_all 20，第三台 flush_all 40，……可能到了第 15 台你就可以执行 flush_all 了。这个要根据具体的情况来实施。

可以通过延时选项来控制在设定的时间之后再使得缓存条目失效。例如：

```
set fll 36 0 2              //设置一个缓存条目 fll
12
```

```
STORED
flush_all 20            //设定 20 秒后缓存条目失效
OK
get fll                //立即查看 fll 条目
VALUE fll 36 2
12
END
get fll                 //20 秒钟后再查看该条目
END
```

2. version 命令

version 命令没有参数：

```
version\r\n
```

服务器发送的响应为：

```
VERSION <version>\r\n
```

这里的<version>是服务器版本的字符串。

例如：

```
version
VERSION 1.4.5
```

3. verbosity 命令

verbosity 命令的功能效果是设置日志输出的级别。它需要指定一个数字作为参数，它的执行总会成功，并且服务器端会返回“OK\r\n”作为响应，除非在该命令的命令行中使用了“noreply”参数。

4. quit 命令

quit 命令没有参数：

```
quit\r\n
```

当服务器端收到该命令后，会关闭这个连接。当客户端不再使用连接时，可以执行该命令来结束一个连接，与此同时，客户端也就简单地关闭了连接，服务器端没有任何消息发送至客户端。

例如：

```
[root@cache ~]# telnet 192.168.3.139 11211
Trying 192.168.3.139...
Connected to cache.xx.com （192.168.3.139） .
Escape character is '^]'.

quit
Connection closed by foreign host.
[root@cache ~]#
```

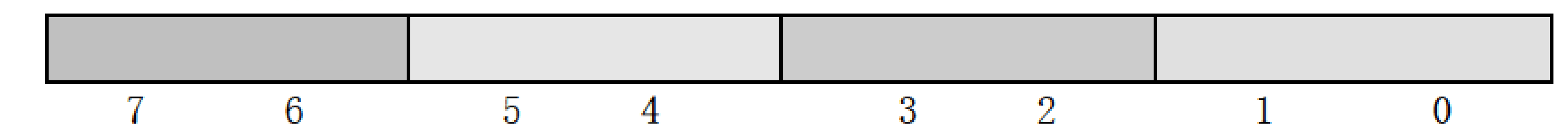
55.6.12 UDP 协议

在一个非常大的网络结构中，基于 TCP 连接的客户端可能会超过 TCP 协议的最大连接数，

当超过这个限制后就不得不启用 UDP 协议。但由于 UDP 协议是基于不可靠的连接，因此，使用 UDP 协议并不能够保证请求的内容被正确返回。具体来说，在使用"get"请求时会得到不完整的响应或者是未能命中缓存，这些都将被看做是缓存失效。

每一个数据包包含一个简单的帧头，跟随帧头的数据具有同样的格式，就像在前边描述的 TCP 协议一样。在当前的执行中，请求必须被包含在一个简单的 UDP 数据包中，但是响应数据包可能会被拆分到多个这样的数据包中。常见请求中，唯一可被拆分成多个数据包的是巨大的多键（multi-key）"get"请求或"set"请求，无论如何，基于可靠性的考虑，这两种操作使用 TCP 连接会更合适。

帧头的长度为 8 个字节。下面是它的结构（帧头中的值都以 16 位即两字节存放，按照网络中字节的顺序，高字节在前）：



- 0～1：请求 ID。
- 2～3：序号。
- 4～5：该信息总共的数据包数量。
- 6～7：保留位，以便将来作为功能扩充使用，当前值必须是 0。

请求 ID 由客户端提供，ID 的典型做法是用一个随机的值，然后单调递增。客户端可以根据自己的喜好自由地选择使用任何请求 ID。在服务器端的返回数据包中包含了同样的 ID，客户端就是通过这个 ID 来区分来自同一台服务器的不同响应，同样服务器端也是通过这个 ID 来区分不同请求的客户端。任何一个未知请求 ID 的数据包，可能是由于响应延时，都将会被丢弃。

序号的范围从 0～n-1，这里的 n 是一个消息被拆分为多个数据包的一个总数，客户端收到所有的数据包后，应该按照这个顺序将所有数据包中的有效载荷连接起来，这样的结果就和使用 TCP 协议的响应格式一样了，结尾同样是一个"\r\n"序列。

55.7 Nginx 的 Memcached 模块

Nginx 提供了 Memcached 模块，通过它来与 Memcached 服务进行操作。在编译安装 Nginx 时，如果没有特别指定"--without-http_memcached_module"选项，那么 Memcached 模块就会被默认安装。先看一下该模块的用法。

1. 配置示例

```
server {
  location / {
set $memcached key $uri;
memcached pass name:11211;
default type   text/html;
error_page 404 @fallback;
  }
```

```
  location @fallback {
proxy_pass backend;
  }
}
```

2. 指令

Memcached 模块提供了以下 7 个命令。

指令名称：memcached_pass

语法：memcached_pass hostname:port

使用环境：location，if

功能：定义 Memcached 守护进程的主机名和端口。

例如：

```
memcached_pass localhost:11211;
```

指令名称：memcached_bind

使用环境：location，if

功能：绑定本地 IP 地址。

例如在本地的 eth0 上绑定了如下的 IP 地址：

78.159.118.168、78.159.118.162 和 78.159.118.163 这三个地址。

而在 server 中做了如下定义：

```
###server1

server {
  listen 78.159.118.168:81;
  listen 78.159.118.162:81;
  listen 78.159.118.162:80;
location / {
proxy pass http://78.159.118.162:8080;
 proxy set header X-Real-IP $remote addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}

###server2

server {
listen 78.159.118.162:8080;
location / {
fastcgi_pass unix:/tmp/fcgi.sock;
fastcgi_param  QUERY_STRING $query_string;
fastcgi_param  REQUEST_METHOD $request_method;
fastcgi_param  CONTENT_TYPE $content_type;
fastcgi_param  CONTENT_LENGTH $content_length;
```

```
    fastcgi_param  HTTP_X_REAL_IP $http_x_real_ip;
    fastcgi_param  HTTP_X_FORWARDED_FOR$proxy_add_x_forwarded_for;
    fastcgi_param  PATH_INFO $uri;
    }
```

这个配置很容易看懂，在访问 server1 的时候，它会去访问 server2，从而实现了代理。但是现在有一个问题，无论客户端请求的 IP 地址是 78.159.118.162 还是 78.159.118.168，而变量$ REMOTE_ADD 的值在 server2 上表现出来的总是 IP 地址 78.159.118.162，那么对于请求 IP 78.159.118.168 的用户，他们能得到想要的结果吗？回答是不可能的！因为 Nginx 在对外连接时使用的 IP 地址是根据某种算法，在众多的 IP 地址中选择排在第一的那个 IP 作为涉外连接。在这种情况下，就需要指令 memcached_bind 来设置了。

指令名称：memcached_connect_timeout

使用环境：locatio，if

功能：定义连接超时，单位为毫秒（默认：60 000）。

例如:

```
memcached_connect_timeout 5000;
```

指令名称：memcached_send_timeout

使用环境：location，if

功能：定义了数据写操作超时，单位为毫秒（默认：60 000）。

例如:

```
memcached_send_timeout 5,000;
```

指令名称：memcached_buffer_size

使用环境：location，if

功能：定义了读和写缓存的大小。

例如:

```
memcached_buffer_size 8k;
```

指令名称：memcached_read_timeout

使用环境：location，if

功能：定义了数据读操作超时，单位为毫秒（默认：60 000）。

例如:

```
memcached_read_timeout 5,000;
```

指令名称：memcached_next_upstream

语法: Values selected among error，timeout，invalid_response，not_found 或 off

默认值：error timeout

使用环境：http，server，location

使用环境：location，if

功能：当指令 memcached_pass 被连接到一个 upstream 区段时（参考 Upstream 模块），该指令会定义匹配条件，只有条件匹配才会跳到下一个 Upstream 模块。

例如：

```
memcached_next_upstream off;
```

3. 变量

memcached 模块提供了 1 个变量。

变量名称：$memcached_key

功能：该变量表示 memcached 中 key 的值。

4. 使用实例

前面说了这么多，但最后表现出来的是配置文件。下面是 Memcached 和 Nginx 结合后的配置文件：

```
[root@cache conf]# cat nginx.conf

...


http {
include   mime.types;
default_type  application/octet-stream;
sendfileon;
keepalive_timeout  65;


server {
  location / {
set $memcached_key $uri;
memcached_pass 192.168.3.139:11211;
memcached_buffer_size 16k;
memcached_read_timeout 30000;
memcached send timeout 30000;
default type   text/html;
error page 404 @fallback;
  }

  location @fallback {
proxy_pass http://192.168.3.139:8080;
  }
}


}
```

配置文件表现出来的内容就是：当用户对 Nginx 发出请求时，它会将请求转发至 Memcached 服务器，当在 Memcached 中找不到相应的 key 时，就会再次请求后端的服务器。下面我们以实例说明一下。

在下面的内容中，我们通过两个例子来说明上面的配置，看一下具体的访问流程。首先我们要明白，Memcached 在 Nginx 之后，而 Tomcat 又在 Memcached 之后：

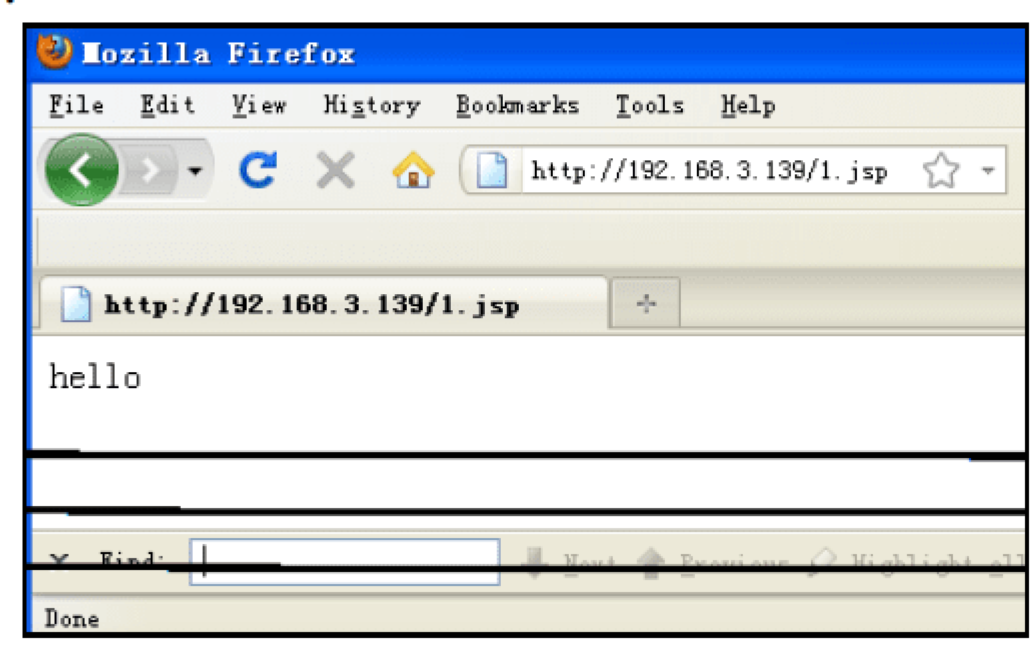
```
Nginx ——> Memcached ——> Tomcat
```

在例子中，目录“/usr/local/tomcat/webapps/ROOT”是 Tomcat 的根目录。

首先编辑一个 1.jsp 文件：

```
[root@cache ROOT]# pwd
/usr/local/tomcat/webapps/ROOT
[root@cache ROOT]# more 1.jsp
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<% out.print ("hello") ;%>
```

然后再访问该网页：



同时监控三种日志：Nginx、Tomcat 和 Memcached。在这里我们看到访问成功，结果是“hello”。

Memcached 的日志：

```
[root@cache bin]# <40 new auto-negotiating client connection
40: going from conn_new_cmd to conn_waiting
40: going from conn_waiting to conn_read
40: going from conn_read to conn_parse_cmd
40: Client using the ascii protocol
<40 get /1.jsp
> NOT FOUND /1.jsp
>40 END
40: going from conn parse cmd to conn mwrite
40: going from conn_mwrite to conn_new_cmd
40: going from conn_new_cmd to conn_waiting
40: going from conn_waiting to conn_read
40: going from conn_read to conn_closing
<40 connection closed.
```

很明显，在缓存中没有“/1.jsp”文件。

Nginx 的访问日志：

```
[root@cache ~]# tail -f /usr/local/nginx0.8.53/logs/access.log
192.168.3.248 - - [27/Jan/2011:14:07:35 +0800] "GET /1.jsp HTTP/1.1" 404
```

```
7 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

同样没有找到。

Tomcat 的访问日志：

```
[root@cache logs]# tail -f localhost access log.2011-01-27.txt
192.168.3.139 - - [27/Jan/2011:14:07:35 +0800] "GET /1.jsp HTTP/1.0" 200 7
```

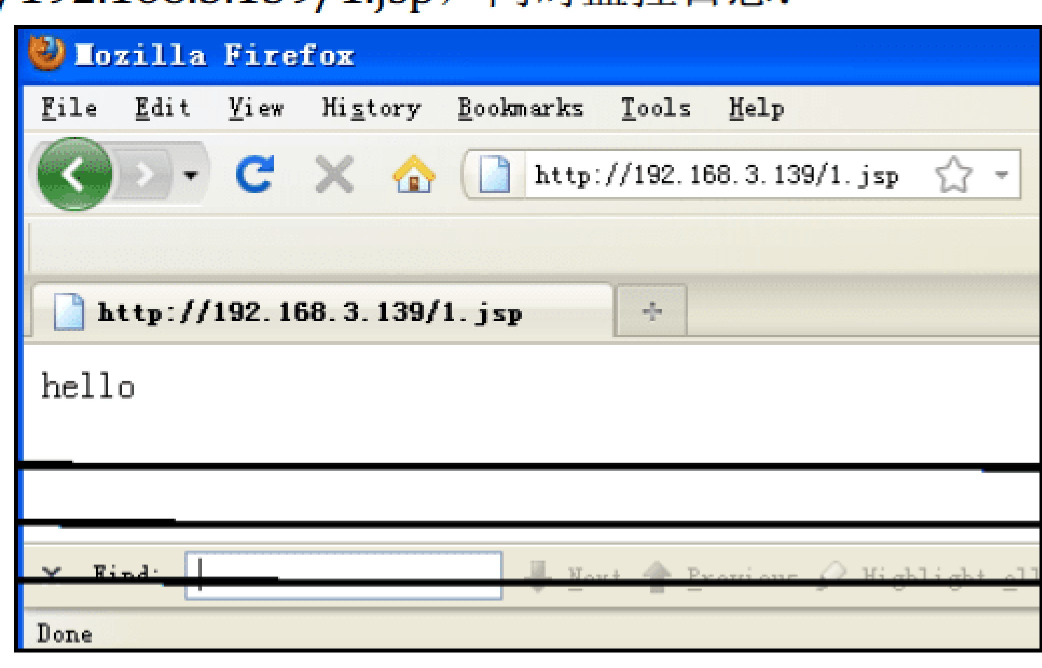
可见在 Tomcat 中访问成功。

由于 Nginx 的模块 memcached 对 Memcached 服务不提供写功能，因此，我们在下面的例子中先手动写入 Memcached 服务。对于程序写入 Memcached 服务器的 Java 客户端很多，例如 Memcached-Java-Client、spymemcached，等等。可以到网址 http://code.google.com/p/memcached/wiki/Clients 上查找，至于使用哪一个客户端，怎样去用，这就不是我们管的了，那是开发人员的事了。

由于 1.jsp 的输出结果是"hello"，因此，我们在 Memcached 服务器中添加一个 key，名字叫"/1.jsp"，而它的值就是"hello"：

```
[root@cache ~]# telnet 192.168.3.139 11211
Trying 192.168.3.139...
Connected to cache.xx.com (192.168.3.139).
Escape character is '^]'.
set /1.jsp 36 0 5
hello
STORED
```

这时再访问 http://192.168.3.139/1.jsp，同时监控日志：



访问成功。下面我们分析一下这个网页来自何处。

下面是 Memcached 服务的日志：

```
[root@cache bin]# <41 new auto-negotiating client connection
41: going from conn new cmd to conn waiting
41: going from conn waiting to conn read
41: going from conn_read to conn_parse_cmd
41: Client using the ascii protocol
<41 get /1.jsp
> FOUND KEY /1.jsp
```

```
>41 sending key /1.jsp
>41 END
41: going from conn_parse_cmd to conn_mwrite
41: going from conn mwrite to conn new cmd
41: going from conn new cmd to conn waiting
41: going from conn waiting to conn read
41: going from conn read to conn closing
<41 connection closed.
```

可见“/1.jsp”文件被找到，而且被发送到客户端。

下面是 Nginx 的访问日志：

```
[root@cache ~]# tail -f /usr/local/nginx0.8.53/logs/access.log
192.168.3.248 - - [27/Jan/2011:14:39:09 +0800] "GET /1.jsp HTTP/1.1" 200
5 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

可见这次的访问状态是 200，表示访问成功。

下面 Tomcat 的访问日志：

```
[root@cache  logs]# tail -f  localhost_access_log.2011-01-27.txt
```

日志没有输出，表明访问就没有到过 Tomcat。

因此，充分说明这个访问的结果是来自于 Memcached 服务器。

测试到这里就算完成任务了，剩下的工作就是程序员的事情了，他们会根据自己的喜好选择一个 Java 客户端来对 Memcached 服务器进行写的操作。在后面章节的 Tomcat 集群中会用到 de.javakaffee.web.msm.MemcachedBackupSessionManager 类，它实现了对 Memcached 服务器的读写。

55.8 Memcached 的客户端

支持 Memcached 服务器的客户端非常多，每一种语言可能还不只一个客户端，例如，C 语言的 libmemcached、PHP 的 memcache 和 memcached（需要 libmemcached 的支持），而 Perl 则提供了更多的客户端，例如，Cache::Memcached、Cache::Memcached::Fast、Memcached::libmemcached、Cache::Memcached::libmemcached，还有其他的就不再列举了，可以参考相关章节。

55.9 libmemcached

libmemcached 是一个 C 语言编写的客户端，C 的效率众所周知，同时其他语言编写的客户端还需要这个库，因此，我们在这里单独认识一下它，另外，单独认识它的原因还有另一个意思，那就是它提供了非常好的 7 条命令。

55.9.1 libmemcached 的安装

```
[root@web1 ~]# wget http://launchpadlibrarian.net/   \
> 33299677/libmemcached-0.29.tar.gz
[root@web1 ~]# tar -zxvf libmemcached-0.29.tar.gz
[root@web1 ~]# cd libmemcached-0.29
[root@web1 libmemcached-0.29]# ./configure
[root@web1 libmemcached-0.29]#make
[root@web1 libmemcached-0.29]#make install
```

看一下 libmemcached 安装后的目录结构：

```
[root@web1 libmemcached-0.29]# tree
.
|-- bin
|   |-- memcat         // 这是 libmemcached 提供的 7 个命令，非常有用
|   |-- memcp          // 对于这些命令后面有说明
|   |-- memerror
|   |-- memflush
|   |-- memrm
|   |-- memslap
|   '-- memstat
|-- include            // 头文件
|   '-- libmemcached
|   |-- memcached constants.h
|   |-- memcached get.h
 …  //省略
|   |-- memcached types.h
|   |-- memcached_util.h
|   '-- memcached_watchpoint.h
|-- lib// 库文件
|   |-- libmemcached.a
|   |-- libmemcached.la
 …  //省略
|   |-- libmemcachedutil.so.0.0.0
|   '-- pkgconfig
|   '-- libmemcached.pc
'-- share              // 相关文档
'-- man
|-- man1               // 相关命令的 man 文档
|   |-- memcat.1
|   |-- memcp.1
 …  //省略
|   '-- memstat.1
'-- man3               // 操作 Memcached 服务器指令的相关说明
```

```
|-- libmemcached.3
|-- libmemcached_examples.3
|-- libmemcachedutil.3
|-- memcached pool destroy.3
  … //省略
|-- memcached stat get keys.3
|-- memcached stat get value.3
|-- memcached_stat_servername.3
|-- memcached_strerror.3
|-- memcached_verbosity.3
'-- memcached_version.3

9 directories, 96 files
```

我们从它的 README 文件中了解到，它就是一个用于连接 Memcached 服务器的 C 库，我们应该了解到各种语言都有单独连接 Memcached 的扩展应用（即客户端），例如，我们刚了解的 PHP 环境下有 Memcache，而 Perl 环境下有 Cache::Memcached，Python 环境下有 python-memcached，而且都不是仅仅一种扩展的实现，而为什么还要使用 libmemcached 库呢？这是因为，它是一个使用 C 语言编写的、高效的、安全的 Memcached 客户端，它超越了所有现有通过独自语言实现的客户端，因此，又有很多开发人员通过具体使用 Memcached 服务器的语言将它“封装”使用，例如上面的 memcached，还有我们在 Perl 部分看到的 Memcached::libmemcached 和 Cache::Memcached::libmemcached 客户端。

另外对于 libmemcached 安装包，也不仅仅是一个单纯的 libmemcached 客户端，除了这个客户端外，我们在前面也看到了，它还包括了 7 条命令，通过这些命令我们可以管理和了解 Memcached 服务器。

好了，下面我们来认识一下这些命令。

55.9.2 命令

在 libmemcached 中，提供了以下 7 条命令，就是前面我们在安装 libmemcached 部分中，安装后的 bin/目录中的指令，它们都非常有用，在下面的篇幅中，我们将分别认识它们。

1. memcp 命令

命令名称：memcp

语法：memcp [options] file file <servers>

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--servers=：指定该命令要连接的服务器，可以同时指定多个。

--flag=：在执行存储操作时提供 flag 信息。

--expire=：为缓存对象设置生存期。

--set：在对对象进行 Memcached 服务器存储时使用 set 指令。

--add：在对对象进行 Memcached 服务器存储时使用 add 指令。

--replace：在对对象进行 Memcached 服务器存储时使用 replace 指令。

--hash=：选择哈希类型。

--binary：切换到二进制协议。

命令格式：

```
[root@web1 bin]# memcp -h
memcp v1.0
```

功能：该命令用于将文件复制到 Memcached 服务器集群上。一次可以复制一个或者多个文件，类似于 Linux 系统中的 cp 命令，memcp 中的 cp 就是从 Linux 中来的，因此它们的功能非常相似。key 的名字将会使用原文件的名字，而且不会带有任何路径（即目录）。在指定 Memcached 服务器时可以使用--servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。如果这两者都没有指定，那么在命令行中最终会使用当前服务器的名称。

使用举例

实例 1：

```
[root@web1 ~]#  memcp   df.pl  --servers=127.0.0.1:11211
[root@web1 ~]#
[root@web1 ~]#  memcp  --debug  df.pl  --servers=127.0.0.1:11211
op: set
source file: df.pl
length: 561
key: df.pl
flags: 0
expires: 0
```

在本实例中我们看到，命令 memcp 实际操作使用的是 set 操作。其他的解释一看便明白。

实例 2：

```
[root@web1~]#memcp--debug--expire=3600df.plcpu.sh--servers=127.0.0.1:11211
op: set
source file: df.pl
length: 561
key: df.pl
flags: 0
expires: 3600
op: set
source file: cpu.sh
length: 223
key: cpu.sh
flags: 0
```

```
expires: 3600
```

在这个例子中，我们同时复制了两个文件，并且设置了生存期。

2. memcat 命令

命令名称：memcat

语法：　memcat [options] key key ...

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--servers=：指定该命令要连接的服务器，可以同时指定多个。

--flag：在执行存储操作时提供 flag 信息。

--hash=：选择哈希类型。

--binary：切换到二进制协议。

命令格式：

```
 [root@web1 bin] # memcat -h
memcat v1.0
```

功能：将 Memcached 服务器中缓存的 key 的值复制到标准的输出设备上。这里的 cat，就是 Linux 系统中 cat 命令的意思。因此，memcat 命令的功能和 Linux 系统的 cat 命令功能相似。在指定 Memcached 服务器时可以使用--servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。

使用举例

实例 1：

```
[root@web1 ~]# memcat cpu.sh  --servers=127.0.0.1:11211
#!/bin/bash
cpuusr='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
cpusys='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
UPtime='/usr/bin/uptime |awk '{print """"}''
echo $cpuusr
echo $cpusys
echo $UPtime
hostname
```

实例 2：

```
[root@web1 ~] # memcat --debug cpu.sh --servers=127.0.0.1:11211
key: cpu.sh
flags: 0
length: 223
value:         #!/bin/bash
cpuusr='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
```

```
cpusys='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
UPtime='/usr/bin/uptime |awk '{print """"}''
echo $cpuusr
echo $cpusys
echo $UPtime
hostname
```

查看文件使用--debug 选项会显示出文件的大小，即长度。

3. memerror 命令

命令名称：memerror

语法：memerror [options] error_code

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

命令格式：

```
[root@web1 bin]  # memerror -h
memerror v1.0
```

功能：将 Memcached 的错误代码转换为可以理解的字符串。

使用举例

实例 1：

```
[root@web1 ~]   # memerror   13
CONNECTION DATA DOES NOT EXIST
[root@web1 ~]   #
[root@web1 ~]   # memerror   15
STORED
[root@web1 ~]   # memerror   16
NOT FOUND
```

可解释的代码。

实例 2：

```
[root@web1 ~]   # memerror   80
Gibberish returned!
You have mail in /var/spool/mail/root
[root@web1 ~]   # memerror   800
Gibberish returned!
```

不可解释的代码。

4. memflush 命令

命令名称：memflush

语法：memflush [options]

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--servers=：指定该命令要连接的服务器，可以同时指定多个。

--expire=：为缓存对象设置生存期。

--binary：切换到二进制协议

命令格式：

```
[root@web1 bin]# memflush  -h
memflush v1.0
```

功能：清除 Memcached 服务器的内容，可以同时指定多台 Memcached 服务器清除。执行此命令的结果意味着所指定的服务器中缓存的内容将被全部清除。在指定 Memcached 服务器时可以使用--servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。

使用举例

实例 1：

```
[root@web1 ~] # memflush --debug --expire=7200 --server=127.0.0.1:11211
[root@web1 ~] # memcat --debug  cpu.sh  --servers=127.0.0.1:11211
key: cpu.sh
flags: 0
length: 223
value:        #!/bin/bash
cpuusr='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
cpusys='/usr/bin/sar -u 1 3 |grep Average |awk '{print }''
UPtime='/usr/bin/uptime |awk '{print """"}''
echo $cpuusr
echo $cpusys
echo $UPtime
hostname
```

实例 2：

```
[root@web1 ~] # memflush  --debug  --server=127.0.0.1:11211
[root@web1 ~] #
[root@web1 ~] # memcat --debug cpu.sh --servers=127.0.0.1:11211
```

5. memrm 命令

命令名称：memrm

语法：memrm [options] key key ...

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--servers=：指定该命令要连接的服务器，可以同时指定多个。

--expire=：为缓存对象设置生存期。

--hash=：选择哈希类型。

--binary：切换到二进制协议。

命令格式：

```
[root@web1 bin]# memrm  -h
memrm v1.0
```

功能：从指定的 Memcached 服务器中移除 key，一次可以移除多个 key。命令 memrm 中的 rm 来自于 Linux 的命令 rm，因此，对于 memrm 来说，与 rm 的功能非常相似。在指定 Memcached 服务器时可以使用 --servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。

使用实例

实例 1：

```
[root@web1 ~]#  memcp --debug df.pl --servers=127.0.0.1:11211
op: set
source file: df.pl
length: 561
key: df.pl
flags: 0
expires: 0
[root@web1 ~]# memrm --debug df.pl --servers=127.0.0.1:11211
key: df.pl
expires: 0
[root@web1 ~]# memcat  --debug df.pl --servers=127.0.0.1:11211
```

实例 2：

```
[root@web1 ~]# memrm --debug df.pl cpu.sh --servers=127.0.0.1:11211
key: df.pl
expires: 0
key: cpu.sh
expires: 0
```

6. memslap 命令

命令名称：memslap

语法：memslap [options]

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--concurrency=：模拟负载的用户数，即实现并发用户。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--execute-number=：执行给定的测试次数，即测试次数。

--flag：在执行存储操作时提供 flag 信息。

--flush：在运行测试前，刷新（即清空所有原有数据）Memcached 服务器。

--help：显示帮助信息并且退出。

--initial-load=：设置在执行测试之前载入的键值（key-value）对的数量。

--non-blocking：设置 TCP 使用非阻塞 IO。

--servers=：列出要连接的服务器。

--tcp-nodelay：这是设置 TCP 套接字不使用底层拼包（即 TCP_NODELAY 选项）。

--test=：设置测试方法，当前支持“get”或者“set”操作。

--version：显示该命令的版本并且退出。

--verbose；详细模式显示。

--binary：切换到二进制协议。

命令格式：

```
[root@mail bin]#  memslap -h
memslap v1.0
```

功能：该命令用于载入测试和基准测试工具，以测试 Memcached 服务器集群的性能。在指定 Memcached 服务器时可以使用 --servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。

使用实例

实例 1：

```
[root@web1~]#memslap--concurrency=280--execute-number=300--server=127.0.0.1
Threads connecting to servers 280
Took 1.713 seconds to load data
```

```
[root@mailbin]#./memslap--concurrency=280--execute-number=300--server=12
7.0.0.1
Threads connecting to servers 280
Took 168.050 seconds to load data
```

不同服务器配置，对于 Memcached 服务器的性能有很大不同。

实例 2：

```
[root@web1~]#memslap--concurrency=280--execute-number=300--server=127.0.
0.1:11211,127.0.0.1:11212
Threads connecting to servers 280
Took 1.653 seconds to load data
```

```
[root@mailbin]#./memslap--debug--concurrency=280--execute-number=300--se
rver=127.0.0.1:11211 --server=127.0.0.1:11212
Threads connecting to servers 280
```

```
Took 189.661 seconds to load data
```

在同一台服务器上运行多个 Memcached 服务器，性能好的机器会提高 Memcached 集群的性能，而性能低的机器却使得 Memcached 集群性能下降。

实例 3：

```
[root@web1 ~]# memstat --server=127.0.0.1:11212

… //省略

 limit_maxbytes: 314572800
 threads: 200
```

调整线程数为 200（这可是个可怕的数）。

```
[root@web1 ~]#memslap --concurrency=280 --execute-number=300 -server
=127.0.0.1:11212

    … //省略

Failured on insert of QYxZ2DaiCxIlfKnY87yCZ4yi7Lljgdm61bX4o5m02wdaHGyPNx
TotRW0hFkLkWRm5zqlwSdrOEHhWFylCSZyn3ySAipu4Hg18Mmw
Failured on insert of JRuJfb2FtoSvjtuz3aQ4iGr3imJC2gALz4gefbtPFEKyz7
y2hOyNYqIFSeJWATrAPOo4raT6gi5fpvZMBXecDwjhQuBdFvfgJRcA
Failured on insert of TS1wvWQOwlvG05j8nZycaw2svBvQhCOAYHyJoOjc2fl2cS2Nu0r
SomCkxz2WCRgccWwOvXtx2ereytusllDPXP9KP4qTXM5zttGo
Failured on insert of KGOQ10IFh807Z3aljOqj0ExwO8cCHLLeTdwFe6MJEMInHkP10Fc
0LaomaOqrdBT7hHEJNcWergP0OitOqWGBwwpWKgDpjbwOsAjG
Failured on insert of yRgsDPcOYaT4Huv4Pd5cYwh8oxDZXPTWkAMybqE7Gb4aTzWu
TRWtN4131ESZYqvG096RzKa8vWaOM4ArXZkmdeiWsAvO0ihsrnJj
Failured on insert of aWI8xese1pSW0a0bOpeRfMcKgzOZyY4P3EP0sHXmyzGyPGPp
63GlHsxyrLNiuRzphOhPZCBX3JLScmAahIByB0ouMBS8WJpBjWd8
Threads connecting to servers 280
Took 12.125 seconds to load data
```

没错，不但时间变成，而且也出现了 Failured。

实例 4：

```
[root@web1 ~]#  memstat   --server=127.0.0.1:11212

… //省略

 bytes_written: 0
 limit_maxbytes: 314572800
 threads: 40
```

调整线程数为 CPU 内核数的 8 倍。

```
[root@web1~]#memslap--concurrency=280--execute-number=300--server=127.0.
0.1:11212
```

```
Threads connecting to servers 280
Took 1.616 seconds to load data
```

可见适当地调整线程数还是有提高的。

7. memstat 命令

命令名称：memstat

语法：memstat [options]

以下为可使用的选项。等号（=）表示该选项需要指定具体的值。

--version：显示该命令的版本并且退出。

--help：显示帮助信息并且退出。

--verbose：详细模式显示。

--debug：调试模式显示，该模式会输出许多有用的信息，对于调试很有用。

--servers=：指定该命令要连接的服务器，可以同时指定多个。

--flag：在执行存储操作时提供 flag 信息。

--analyze=：分析提供的服务器。

命令格式：

```
[root@web1 bin]# memstat -h
memstat v1.0
```

功能：显示单个或者是一组（即多个）Memcached 服务器中的操作状态，将所有信息显示输出到标准的输出设备上。看上去像 Linux 系统的 stat 命令，但是功能完全不同。在指定 Memcached 服务器时可以使用 --servers 选项，也可以使用“MEMCACHED_SERVERS”环境变量。

使用实例

实例 1：

```
[root@web1 ~]# memstat  --server=127.0.0.1:11211
Unknown key auth cmds
Unknown key auth errors
Unknown key conn_yields
Unknown key reclaimed
Listing 1 Server

Server: 127.0.0.1 (11211)
 pid: 10369
 uptime: 4231192
 time: 1315550431
 version: 1.4.5
 pointer_size: 32
 rusage_user: 11.184299
 rusage system: 23.980354
 curr_items: 14
```

```
 total_items: 295308
 bytes: 19089
 curr_connections: 5
 total connections: 6796
 connection structures: 18
 cmd get: 468458
 cmd set: 295309
 get_hits: 294741
 get_misses: 173717
 evictions: 0
 bytes_read: 19089
 bytes written: 19089
 limit maxbytes: 314572800
 threads: 4
```

实例 2：

```
[root@web1 ~]# memstat --server=127.0.0.1:11211,127.0.0.1:11212
Unknown key auth_cmds
Unknown key auth errors
Unknown key conn yields
Unknown key reclaimed
Unknown key auth cmds
Unknown key auth_errors
Unknown key conn_yields
Unknown key reclaimed
Listing 2 Server

Server: 127.0.0.1 (11211)
 pid: 3852
 uptime: 4904
 time: 1315557959
 version: 1.4.5
 pointer_size: 32
 rusage user: 6.175061
 rusage system: 10.586390
 curr items: 404408
 total items: 404633
 bytes: 222839930
 curr_connections: 5
 total_connections: 1614
 connection_structures: 285
 cmd get: 354
 cmd set: 404633
 get_hits: 206
```

```
    get_misses: 148
    evictions: 0
    bytes_read: 222839930
    bytes written: 222839930
    limit maxbytes: 2097152000
    threads: 4

   Server: 127.0.0.1 (11212)
    pid: 10523
    uptime: 928
    time: 1315557958
    version: 1.4.5
    pointer size: 32
    rusage user: 2.547612
    rusage system: 4.488317
    curr_items: 171000
    total_items: 171000
    bytes: 94221000
    curr connections: 41
    total connections: 616
    connection structures: 331
    cmd get: 0
    cmd_set: 171000
    get_hits: 0
    get_misses: 0
    evictions: 0
    bytes read: 94221000
    bytes written: 94221000
    limit maxbytes: 314572800
    threads: 40
```

55.9.3 函数

libmemcached 提供了相当多的函数，并且不同的版本函数也不同，以下是 0.32 版本下的函数。对于函数的使用方法，就不多讲了，毕竟我们是做运维的，不是搞开发的，如果你确实对这些函数感兴趣，那么可以去 man。

libmemcached	memcached_fetch_execute
memcached_replace_by_key	memcached_fetch_result
libmemcached_examples	memcached_flush_buffers
memcached_server_add	memcached_free
libmemcachedutil	memcached_generate_hash_value
memcached_server_count	memcached_get
memcached_add	memcached_get_by_key

memcached_server_list	memcached_get_memory_allocators
memcached_add_by_key	memcached_get_user_data
memcached_server_list_append	memcached_increment
memcached_analyze	memcached_increment_with_initial
memcached_server_list_count	memcached_lib_version
memcached_append	memcached_mget
memcached_server_list_free	memcached_mget_by_key
memcached_append_by_key	memcached_pool_create
memcached_server_push	memcached_pool_destroy
memcached_behavior_get	memcached_pool_pop
memcached_servers_parse	memcached_pool_push
memcached_behavior_set	memcached_prepend
memcached_set	memcached_prepend_by_key
memcached_callback_get	memcached_quit
memcached_set_by_key	memcached_replace
memcached_callback_set	memcached_set_user_data
memcached_set_memory_allocators	memcached_cas_by_key
memcached_cas	memcached_stat
memcached_version	memcached_clone
memcached_dump	memcached_stat_get_keys
memcached_fetch	memcached_create
memcached_decrement_with_initial	memcached_stat_get_value
memcached_strerror	memcached_decrement
memcached_delete	memcached_stat_servername
memcached_verbosity	memcached_delete_by_key

下面简单地举一个例子。

函数名称：memcached_increment，memcached_decrement

功能：Memcached 服务器能够增加或者是减少键的值（对上溢和下溢不做检查）。

memcached_increment()：该函数用于增加值（就是我们所说 key-value 中的 value）的长度，它通过获取 offset 传递的值来控制增加值的长度，我们通过它的函数代码可以看出，offset 是一个无符号的整数。

memcached_decrement()：该函数正好与 memcached_increment()相反，它是用于缩短值长度的，同样它也是通过获取 offset 传递的值来控制增加值的长度。

这两个函数的返回值类型是“memcached_return”，访问成功，那么返回值为“MEMCACHED_SUCCESS”，使用 memcached_strerror()函数可以将这个值转换为一个可打印的字符串。

摘要：

```
#include <memcached.h>

memcached_return
  memcached_increment （memcached_st *ptr,
```

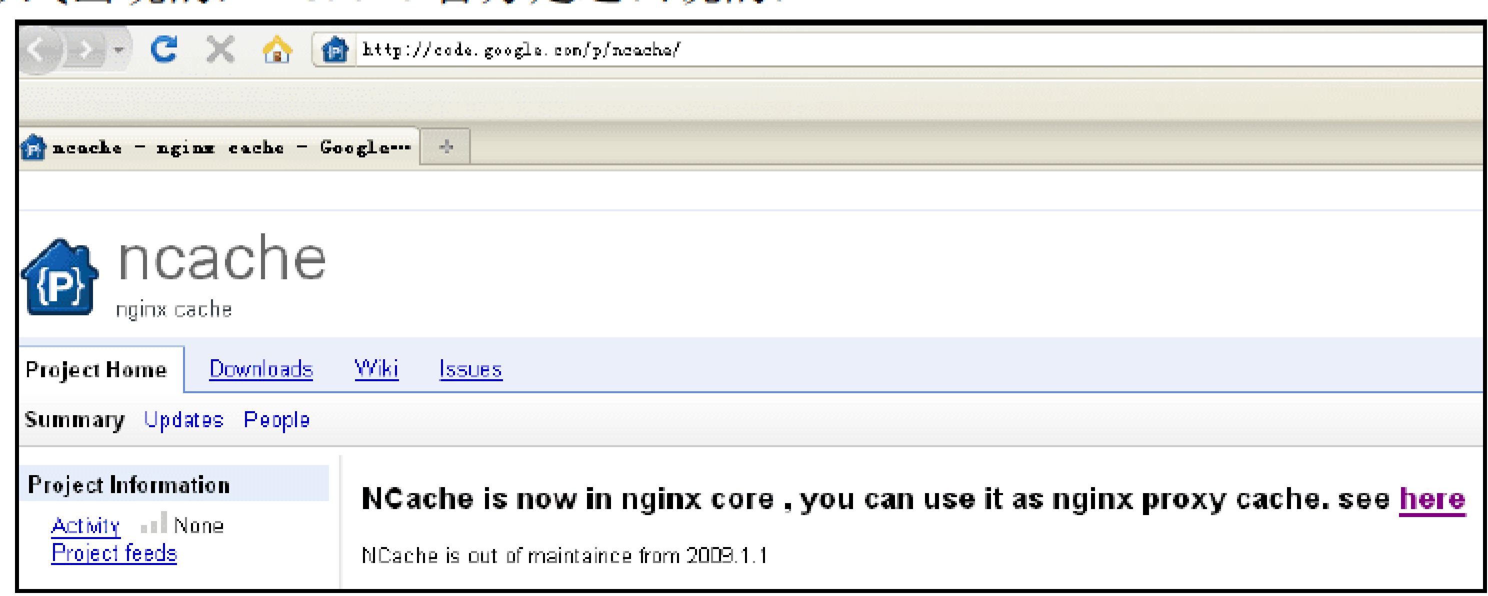
```
    const char *key, size_t key_length,
    unsigned int offset,
    uint64_t *value) ;

     memcached return
       memcached_decrement （memcached_st *ptr,
    const char *key, size t key length,
    unsigned int offset,
    uint64_t *value) ;
```

第 56 章　缓存技术——NCache

NCache 缓存方式我们只需要了解一下就可以了。

第三方模块用得较多的就是新浪网的开源项目——NCache。这是一个比较古老的模块了，它只支持 Nginx 的 0.6.x 版本，对于 Nginx 的其他版本并不支持，相反的是在后面的版本中是以 Nginx 内核形式出现的，NCache 官方是这么说的：



但是为了理解该模块，我们还是以 Nginx 0.6.39 和 NCache 2.3 讲述一下。下面的这个例子是比较早期的一个应用。NCache 现有的版本是 3.1 和 2.3，其中 3.1 只能用在 64 位的 Linux 系统上，由于当时在我们的环境中没有 64 位的 Linux，所以就使用了 2.3 版。在下面的讲述中我们同样会阐述 3.1 的使用。下面一段文字来自于 NCache 的 wiki：

NCache 是基于 Nginx 的 Web 服务器模型构建起来的缓存系统，是 SINA 公司的开源产品。

起初的目的是为了提升缓存响应速度而开发的，因为 Squid 实在太慢，而 Nginx 的优势就在于网络服务上，所以 NCache 计划也就诞生了。

NCache 最早的时候是作为 Nginx 的一个 HTTP 模块进行开发的，因为当时希望实现更好的兼容性和可扩展性，作为独立模块，可以被更好地推广和使用，安装也会很方便。但后来发现，随着代码量的增加、功能的扩充，Nginx 的原有模块框架已经不能很好地满足我们了，因此，我们提取了 Nginx 的内核代码，并把 Cache 部分嵌入其中，形成了今天的 NCache。

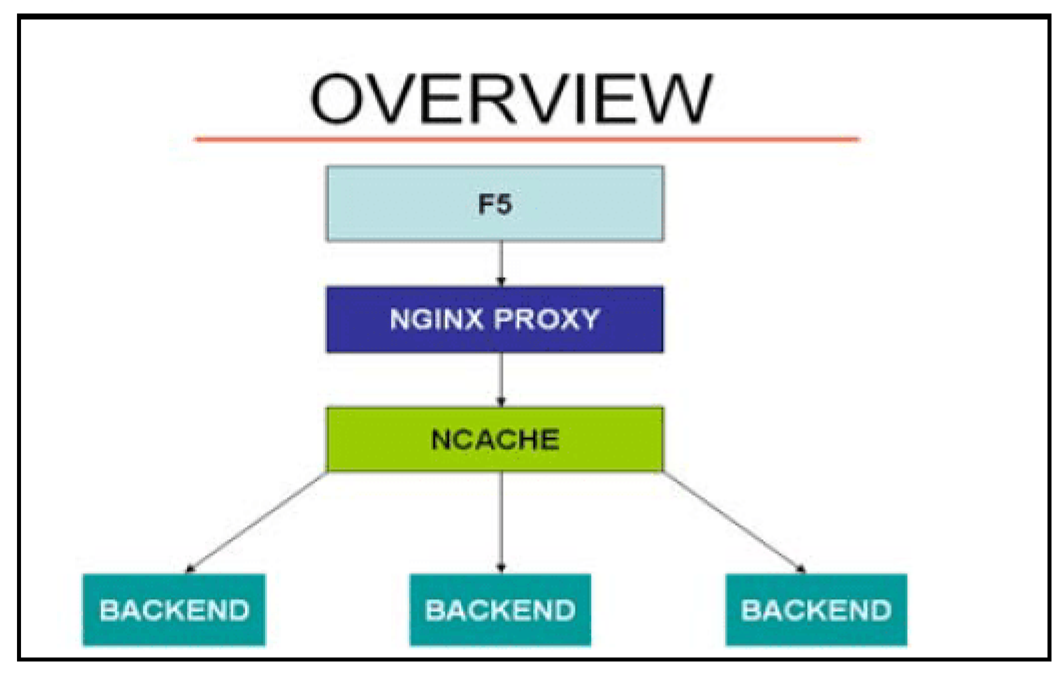
NCache 本身功能并不强大，且不具备像 SQUID 般完善的功能和开发框架，甚至不能支持 RFC 中关于 Cache 部分的描述。NCache 完全是一套定制化的产品，可以满足快速部署，简单易用、大并发量，是大存储量朋友们的需求，它不需要复杂的配置，不需要冗余的复杂代码，并使用最先进的技术组合。

NCache 2.0 版本，是作为一个完整的 Nginx 模块进行发布和使用的，从原有的 NCache 内核中进行了剥离，更加方便开发者的安装和配置。

NCache 3.0 版本，相对于 2.0 版本有了很大的改进，对文件的缓存不再使用传统的目录模式，而是通过 MMAP 一个大文件，在其中以页分配的形式存储缓存数据，由操作系统来负责决定哪些数据应该留在内存中。这与 Varnish 缓存的原理是一致的，大大提高了 IO 性能。目前该版本只支持 64 位 Linux 和 Freebsd 系统。

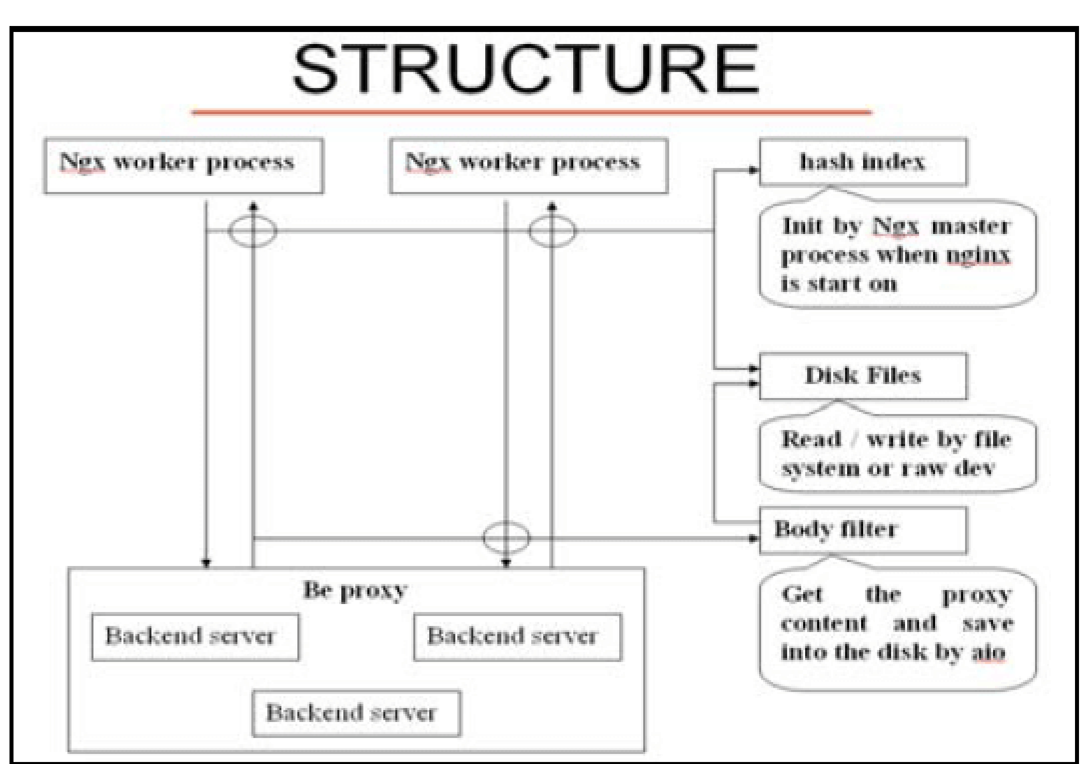
56.1 NCache 工作层示意图

NCache 工作层示意图：



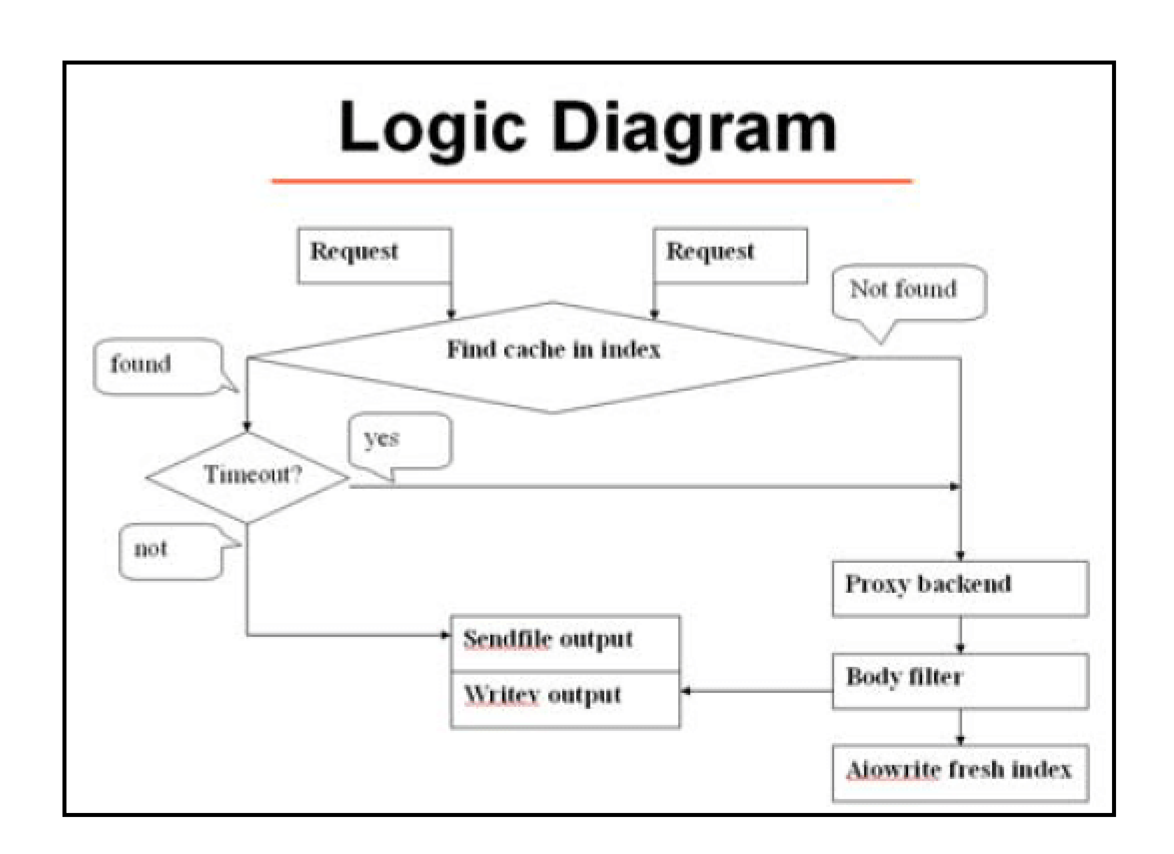
如果没有 F5，也可以使用 LVS，在我所运营的环境中就是使用了 LVS。

缓存原理结构图：



56.2 请求逻辑图

请求逻辑图：



56.3 安装 NCache

```
[root@cache ~]# wget http://nginx.org/download/nginx-0.6.39.tar.gz
[root@cache ~]# wget http://ncache.googlecode.com/files/ncache
-2.3 release.tar.gz
[root@cache ~]# tar -zxvf  ncache-2.3 release.tar.gz
ncache-2.3/
ncache-2.3/config
ncache-2.3/ncache_http_get_cache_module.c
[root@cache ~]# tar -zxvf  nginx-0.6.39.tar.gz
[root@cache ~]# mv  ncache-2.3  nginx-0.6.39/src/http/modules/

[root@cache nginx-0.6.39]# ./configure --prefix=/usr/local/nginx-0.6.39  \
                          --with-pcre=/root/pcre-8.01  \
                          --with-openssl=/root/openssl-0.9.8e  \
                          --add-module=/root/ncache-2.3
```

由于这种方式已经不再流行使用，而该模块在 Nginx 0.7.44 版后将其加入 Nginx 内核，及新加入的 proxy cache 功能。因此在这里就不再讲述它的使用了。我原来运维的几台使用该模块的机器也已经升级了，当时也没把这个东东当做回事就没留下文本资料，之后就改用 Memcached 作为缓存了，当然还有 Varnish。因此在这里只将官方文档中的两个配置文件附在下文：

56.4 配置文件

1. 2.3 版本配置文件

```
user www www;

worker processes 4;

worker rlimit nofile 20480;

error_log  logs/error.log;

events
{
 use epoll;

worker connections 81920;
}

http
```

```
    {
    keepalive_timeout 1800;

    ncache max size 24;

    proxy buffering off;

    ncache_dir /data1/ngx_cache/ 128 64;
    ncache_dir /data2/ngx_cache/ 128 64;
    ncache_dir /data3/ngx_cache/ 128 64;
    ncache_dir /data4/ngx_cache/ 128 64;

    ncache ignore client no cache on;

    upstream backend
     {
    server 10.0.0.1;
       }

    sendfile on;

    send timeout 90;

    client_header_timeout 120;

    tcp_nodelay on;

     log format main'$proxy add x forwarded for - $remote user[$time local] '
       '"$request" $status $bytes sent '
    '"$http referer" "$http user agent" $remote addr';
      include "mime.types";

    server
    {
    server name  .blog.sina.com.cn;

    listen *:80;

    set $xvia "blog.sina.com.cn";

     set $proxy_add_agent "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
NginxCache) ";
    access_log logs/blog.sina.com.cn-access_log main;
```

```
 location /
{

 if ($request_method ~ "PURGE")
 {
  rewrite (.*) /PURGE$1 last;
 }
if ($request_uri = /)
{
rewrite ^/$ /lm/index.html last;
}

ncache http cache;

error_page  404 = /fetch$request_uri;

add_header  Sina-Cache   $xvia;

}

location /ncache state
{
ncache_state;
}


 location /fetch
{
  internal;
 proxy_pass http://backend;
 add_header  Sina-Cache   $xvia;
 proxy_hide_header   User-Agent;
 proxy set headerUser-Agent $proxy add agent;
 proxy set headerX-Forwarded-For $proxy add x forwarded for;
}

   location /PURGE/
{
access_log logs/purge.blog.sina.com.cn-access_log main;
internal;
allow 10.55.37.0/24;
allow 10.69.3.0/24;
```

```
allow 10.49.10.0/24;
denyall;
ncache_purge;
}

}

}
```

2. 3.1 版本配置文件

```
user www www;

worker processes 4;

worker_rlimit_nofile 20480;

error_log  logs/error.log;

events
{
use epoll;

worker_connections 81920;
}

http
{
keepalive timeout 10;

ncache_max_size 24;

proxy_buffering off;

ncache dir /data1/cache file  60;#the file for storage,60G
ncache dir /data2/cache file  60;
hash index dir /data1/ncache index;#v3.1,you need to define the index file
position
auto_delete_file on;#enable the auto delete file
ncache_ignore_client_no_cache on;

upstream backend
{
server 10.0.0.1;
}
```

```
    sendfile on;

    send timeout 10;

    client header timeout 10;


    tcp_nodelay on;

    log_format main  '$proxy_add_x_forwarded_for - $remote_user [$time_local] '
     '"$request" $status $bytes sent '
     '"$http referer" "$http user agent" $remote addr';

    include "mime.types";

    default_type text/html;

    server
    {
    server name  .blog.sina.com.cn;

    listen *:80;

    set $xvia "blog.sina.com.cn";

    set $proxy add agent "Mozilla/4.0  (compatible; MSIE 6.0; Windows NT 5.1;
NginxCache) ";
    access log logs/blog.sina.com.cn-access log main;

    location /
    {

    if  ($request method ~ "PURGE")
    {
    rewrite  (.*)  /PURGE$1 last;
    }


    ncache_http_cache;

    error page  404 = /fetch$request uri;
```

```
add_header  Sina-Cache   $xvia;

}

location /ncache state
{
ncache state;
}


location /fetch
{
internal;
proxy pass http://backend;
add header  Sina-Cache   $xvia;
proxy_hide_header   User-Agent;
proxy_set_headerUser-Agent $proxy_add_agent;
proxy_set_headerX-Forwarded-For $proxy_add_x_forwarded_for;
}


location /PURGE/
{
access_log logs/purge.blog.sina.com.cn-access_log main;
internal;
allow 10.55.37.0/24;
allow 10.69.3.0/24;
allow 10.49.10.0/24;
denyall;
ncache purge;
}

}

}
```

还需要提醒的一点是，如果你真要使用这种架构，那么在后端的服务器配置中必须有生存期设置，如果没有对缓存的内容设置缓存生存期，那么 NCache 不会缓存内容。

第 57 章　缓存技术——Varnish

在 Nginx 中 Varnish 缓存是通过代理模块来实现的，在我使用的 Varnish 缓存服务器中，其中之一就是为了 Apache 而使用它，我们将会在下面的内容中看到。

Varnish 是一个高性能的、先进的 Web 加速器，可以安装在 Linux 2.6、FreeBSD 6/7 和 Solaris 10 系统上，以便发挥其高效的性能，而且是一款开源软件，Varnish 的官方网站为 www.varnish-cache.org。

我们在生产环境中使用了它，就是看中它的加速性能，将它使用在 Nginx 和 Apache 之间，即形成了以下格式：

```
    Client  <-----> Nginx <-----> Varnish <-----> Apache
```

没有将 Apache 改为 Nginx 的原因在于嫌麻烦，懒得去移植而已。

57.1　了解 Varnish

我们首先了解一下 Varnish 的安装条件及其功能。

1. 安装 Varnish 的前提条件

为了安装 Varnish，必须具备以下条件。

- 系统要求：现代版的 64 位 Linux、FreeBSD 和 Solaris；
- 访问权限：对系统具有 root 访问的访问权限。

Varnish 也可以安装在其他 UNIX 系统，但是在这些平台上没有测试过性能。

- 32 位的 Linux、FreeBSD 和 Solaris；
- OS X；
- NetBSD；
- OpenBSD。

2. Varnish 的功能

下面是 Varnish 的一些功能：

- 现代的设计；
- VCL：一个非常灵活的配置语言；
- 后端服务器负载均衡与健康检查；
- 部分支持 ESI（Edge Side Includes）；
- URL 重写；
- 优美地处理“死”后端。

将来的功能（实验性）：

- 支持 Ranged 头；

- 支持持续缓存。

3. 安装要求

操作系统方面，我们在前面说了，可以是 Linux 2.6、FreeBSD 6/7 和 Solaris 10 系统，在这里我选择了 Linux 2.6，在硬件方面，则依赖于我们的实际情况，内存可以使用 1～16GB，磁盘可以选择 SSD，而对于主板，我们可以定制，但是能够支持上 T 内存的主板还是很罕见的，因此，如果缓存较大使用 SSD 磁盘还是一个选择。但是对于大多数用户还是选择品牌服务器，或者是自己攒的机器，只要性能高且稳定就可以了。还有一点要说的是，网卡要选择千兆网卡，否则你也没必要建立 Varnish 缓存，对吗？

对于 32 位系统，Varnish 能够控制的内存不会超过 2GB，因此，如果在高访问量的生产环境下，还是使用 64 位操作系统。

4. 安装 Varnish

关于安装，可选择的方案有两种，一种就是使用 rpm 包（我们主要阐述的是在 Red Hat 下的安装），另一种就是使用从源代码编译，这也是我们要选择的方式。首先通过使用 svn 命令获取源代码。

> 到目前为止 Varnish 的稳定版本为 2.1.5，但最新的版本已经是 3.0 beta 1，估计用不了多久就发布正式版了，因此，我们将会以 3.0 beta 1 为例安装，如果在你使用 Varnish 时仍然没有发布最新的稳定版，那么继续使用稳定版本 2.1.5（Varnishd 在今年的 6 月 17 号正式推出了 3.0.0 稳定版）。

在安装之前需要确定一下需要的工具，这里以 Red Hat 为例：

- automake
- autoconf
- libtool
- ncurses-devel
- libxslt
- groff
- pcre-devel
- pkgconfig

看一下我们的系统：

```
[root@cache ~]# uname -a
Linux cache 2.6.25 #3 SMP Wed Apr 21 21:22:31 CST 2010 i686 i686 i386 GNU/Linux
```

下载并编译安装：

```
[root@cache ~]#wget http://repo.varnish-cache.org/source/varnish
-3.0.0-beta1.tar.gz
[root@cache ~]# tar -zxvf varnish-3.0.0-beta1.tar.gz
[root@cache ~]# cd varnish-3.0.0-beta1
```

看一下源代码目录：

```
[root@cache varnish-3.0.0-beta1]# tree -L 2
.
|-- ChangeLog
|-- INSTALL
|-- LICENSE
|-- Makefile.am
|-- Makefile.in
|-- README
|-- aclocal.m4
|-- autogen.sh
|-- bin                    //安装后的 bin 目录
|   |-- Makefile.am
|   |-- Makefile.in
|   |-- varnishadm
|   |-- varnishd           //该目录被安装在 sbin 目录下，要注意这个问题
|   |-- varnishhist
|   |-- varnishlog
|   |-- varnishncsa
|   |-- varnishreplay
|   |-- varnishsizes
|   |-- varnishstat
|   |-- varnishtest
|   '-- varnishtop
|-- config.guess
|-- config.h.in
|-- config.sub
|-- configure
|-- configure.ac
|-- depcomp
|-- doc                     //网页形式的帮助文件
|   |-- Makefile.am
…
|   |-- changes.html
|   |-- changes.rst
|   '-- sphinx
|-- etc                     //安装后的 etc 目录
|   |-- Makefile.am
|   |-- Makefile.in
|   |-- default.vcl
|   '-- zope-plone.vcl
|-- include                  //源代码的头文件
|   |-- Makefile.am
```

```
…
|   |-- vsm.h
|   '-- vss.h
|-- install-sh
|-- lib                           //安装后的 lib 目录库文件目录
|   |-- Makefile.am
…
|   |-- libvgz
|   '-- libvmod_std
|-- ltmain.sh
|-- m4
|   |-- ax pthread.m4
|   |-- libtool.m4
|   |-- ltoptions.m4
|   |-- ltsugar.m4
|   |-- ltversion.m4
|   '-- lt~obsolete.m4
|-- man                           //man 文档
|   |-- Makefile.am
|   |-- Makefile.in
|   |-- varnish-cli.7
|   '-- vcl.7
|-- missing
|-- redhat                        //注意这个目录，在后面会详述
|   |-- Makefile.am
|   |-- Makefile.in
|   |-- README.redhat
|   |-- TODO
|   |-- varnish.initrc
|   |-- varnish.logrotate
|   |-- varnish.spec
|   |-- varnish.sysconfig
|   |-- varnish_reload_vcl
|   |-- varnishlog.initrc
|   '-- varnishncsa.initrc
'-- varnishapi.pc.in

27 directories, 146 files
```

编译安装：

```
[root@cachevarnish-3.0.0-beta1]#./configure--prefix=/usr/local/varnish-3.0.0-beta1
[root@cache varnish-3.0.0-beta1]#make
[root@cache varnish-3.0.0-beta1]#make install
```

```
…
----------------------------------------------------------------------
Libraries have been installed in:
   /usr/local/varnish-3.0.0-beta1/lib/varnish

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
 during execution
   - add LIBDIR to the 'LD RUN PATH' environment variable
 during linking
   - use the ''-Wl,-rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld (1)  and ld.so (8)  manual pages.
----------------------------------------------------------------------

…
```

在安装过程中可能会出现以下问题：

```
[root@s8 varnish-2.1.5]# uname -a
Linux s8.xx.cn 2.6.9-5.ELsmp #1 SMP Wed Jan 5 19:30:39 EST 2005 i686 i686
i386 GNU/Linux
[root@s8 varnish-2.1.5]#
[root@s8 varnish-2.1.5]# ./configure --prefix=/usr/local/varnish-2.1.5

…

checking for library containing initscr... -lcurses
checking for library containing pthread create... -lpthread
checking for socket in -lsocket... no
checking for getaddrinfo in -lnsl... yes
checking for cos in -lm... yes
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking for PCRE... no
configure: error: Package requirements  (libpcre)  were not met:

Package libpcre was not found in the pkg-config search path.
Perhaps you should add the directory containing 'libpcre.pc'
```

```
to the PKG_CONFIG_PATH environment variable
No package 'libpcre' found

Consider adjusting the PKG_CONFIG_PATH environment variable if you
installed software in a non-standard prefix.

Alternatively, you may set the environment variables PCRE_CFLAGS
and PCRE_LIBS to avoid the need to call pkg-config.
See the pkg-config man page for more details.
```

根据提示可以添加以下参数再次执行：

```
[root@s8 varnish-2.1.5]# ./configure --prefix=/usr/local/varnish-2.1.5 \
> PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

看一下安装后的目录结构：

```
[root@ cache varnish-3.0.0-beta1]# tree
.
|-- bin
|   |-- varnishadm
|   |-- varnishhist
|   |-- varnishlog
|   |-- varnishncsa
|   |-- varnishreplay
|   |-- varnishsizes
|   |-- varnishstat
|   |-- varnishtest
|   '-- varnishtop
|-- etc
|   '-- varnish
|   '-- default.vcl
|-- include
|   '-- varnish
|   |-- varnishapi.h
|   |-- vsc.h
|   |-- vsc all.h
|   |-- vsc fields.h
|   |-- vsl.h
|   |-- vsl_tags.h
|   '-- vsm.h
|-- lib
|   |-- libvarnishapi.a
|   |-- libvarnishapi.la
|   |-- libvarnishapi.so -> libvarnishapi.so.1.0.0
|   |-- libvarnishapi.so.1 -> libvarnishapi.so.1.0.0
|   |-- libvarnishapi.so.1.0.0
```

```
|   |-- pkgconfig
|   |   '-- varnishapi.pc
|   '-- varnish
|   |-- libvarnish.a
|   |-- libvarnish.la
|   |-- libvarnish.so
|   |-- libvarnishcompat.a
|   |-- libvarnishcompat.la
|   |-- libvarnishcompat.so
|   |-- libvcl.a
|   |-- libvcl.la
|   |-- libvcl.so
|   |-- libvgz.a
|   |-- libvgz.la
|   |-- libvgz.so
|   '-- vmods
|   |-- libvmod_std.a
|   |-- libvmod_std.la
|   '-- libvmod_std.so
|-- sbin
|   '-- varnishd
|-- share
|   '-- man
|   |-- man1
|   |   |-- varnishadm.1
|   |   |-- varnishd.1
|   |   |-- varnishhist.1
|   |   |-- varnishlog.1
|   |   |-- varnishncsa.1
|   |   |-- varnishreplay.1
|   |   |-- varnishsizes.1
|   |   |-- varnishstat.1
|   |   |-- varnishtest.1
|   |   '-- varnishtop.1
|   '-- man7
|   |-- varnish-cli.7
|   '-- vcl.7
'-- var
'-- varnish

16 directories, 51 files
```

- bin/：该目录下是 Varnish 提供的工具命令。
- etc/：配置文件所在的目录，varnish/default.vcl 为默认的配置文件。有关该文件的配置

我们将会在 VCL 部分讲述。

- lib/：该目录包含了所有的库文件。
- sbin/：该目录包含了唯一的一个命令，该命令用来运行 Varnish 服务器。
- share/：man 文档所在的目录。
- var/：在 Varnish 服务器没有运行之前该目录下的 varnish 目录为空，但在 Varnish 服务器启动之后，该目录会有类似以下的内容：

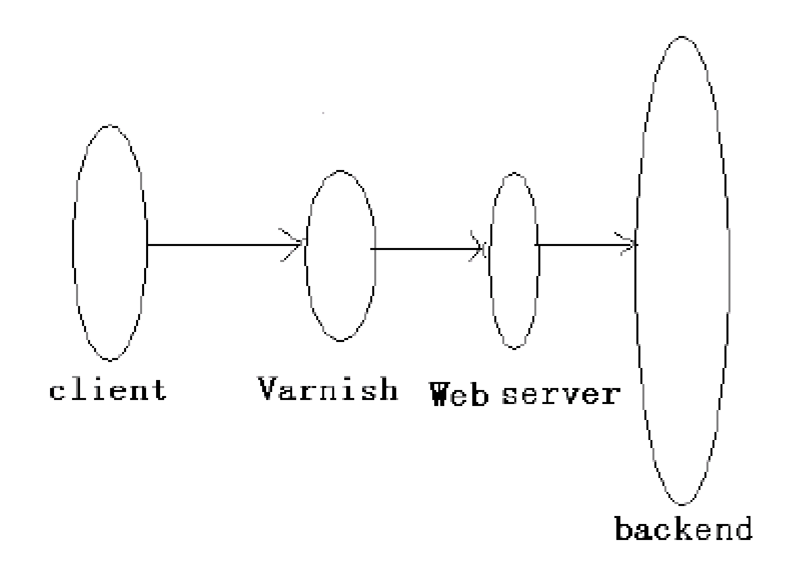
```
[root@ cache var]# tree
.
'-- varnish
'-- cache                    //这里的 cache 来源于服务器的名字
|--  .vsm
'-- vcl.1P9zoqAU.so

2 directories, 2 files
```

57.2 Varnish 的访问部署

我们在测试时将 Varnish 监听的端口设置为 8080，但在具体的使用中往往是监听在 80 端口，因此在这里我们有必要分析一下 Varnish 服务器在网站部署时所处于的位置。在以往的环境中，我们一般是将其作为反向代理来进行缓存，以 80 端口对外提供 HTTP 请求访问，而在使用 Nginx 的网站部署中却往往不是这样，而是让 Nginx 作为反向代理，让 Varnish 作为缓存服务器，因此就会有以下两种不同的部署方案。

57.2.1 第一种部署方案：Varnish 提供 80 访问



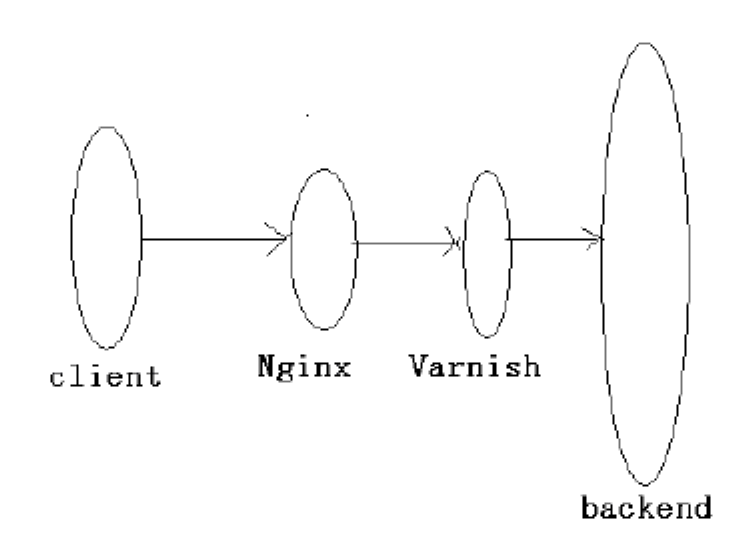
在这种部署中，Varnish 监听在 80 端口，它对外提供 HTTP 访问，在处理客户提交的请求时，首先会查找自己缓存中是否有客户端访问的内容，如果有则提供，如果没有它会向后面的 Web 服务器提出访问，因此要明白的是，Varnish 只是一个缓存，它并不做解析处理，只是作为一个客户端向后台的服务器发出请求，取回客户端需要访问的内容。这里的 Web 服务器既可以是 Nginx、Apache，也可以是其他的 Web 服务器，它们就是在 Varnish 配置文件中被配置为 backend 的主要成分，在这些服务器之后通常还会有其他的服务器，例如，可能会有处理 JSP 的 Tomcat，等等，在这个示意图中我们称为 backend，它和 VCL 中的 backend 是不一样的，因此不要混淆。

如果 Web 服务器运行在同一台物理服务器上，那么还需要将 Web 服务器的端口修改为非 80 端口，然后再将 Varnish 配置文件中的后台服务器中的端口改为 Web 监听的端口。然后再启动 Varnish 服务器：

```
[root@cache ~] varnishd -f /usr/local/varnish-3.0.0-beta1/etc/varnish
/default.vcl -s malloc,1G -T 127.0.0.1:2000
```

在前面我们了解过，省略了-a 参数，则表示监听在 80 端口。

57.2.2 第二种部署方案：Varnish 位于 Nginx 之后只提供缓存



从上面的图中可以看出，由 Nginx 处理客户端的 HTTP 请求，Nginx 会在自己的配置中查询，如果是属于缓存的内容，那么它就会向后台的 Varnish 索取，如果 Varnish 缓存中有客户端的请求，那么 Varnish 就会提供，如果没有，Varnish 就会向后端的服务器发出 HTTP 请求。这里的后台服务器就是图中的 backend，而这个 backend 之后还会有其他的服务器，在此并没有画出。

在上面的两种结构中，虽然图中标示为客户端访问，但在这里绝对不要理解为这是客户端的全部访问，它可能是众多请求中的一个请求，也许是静态的 html 页面、图片文件或者是 css 等。另外这些服务器，Varnish、Web 服务器（包括 Apache、Nginx 等）和后端服务器（包括 Apache、Nginx、Tomcat 等），既可能是在同一台物理的服务器上，也可能是在多台机器上。

> 另外一点要说的是，尽管 Varnish 支持 80 端口正常的 HTTP 访问，但是由于 Nginx 的高并发处理，因此我们最好使用第二种部署方案。

57.3 Nginx 与 Varnish 的结合

选定了部署方案后，我们了解一下 Nginx 与 Varnish 的结合情况。看下面的配置文件：

```
server {

listen   80;
server_name  www.xx.cn;
#  ssi on ;
index index.shtml index.htm index.php;

location / {
   proxy_pass  http://192.168.4.60:80/;
```

```
    }

  …

  }
```

这是一个简化的 Nginx 配置。在这个配置文件中，192.168.4.60 便是 Varnish 服务器，而 Varnish 服务器之后就是 Apache 服务器。原理非常简单，通过代理来实现缓存访问。

57.4 针对 Linux 系统设置

由于 Varnish 本身的特点，因此要根据实际情况对 Linux 系统进行优化和限制使用资源的设置。

57.4.1 Linux 优化内核

这一步必不可少，如果不进行内核的优化，那么性能将达不到最优。主要是对 TCP/IP 部分的优化。看下面的配置：

```
[root@s8 bin]# cat /etc/sysctl.conf

…

net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_time = 180
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp tw reuse = 1
net.ipv4.tcp tw recycle = 1
net.ipv4.ip_local_port_range = 350065000
```

需要执行以下命令，以便立刻生效：

```
[root@s8 bin]#sysctl -p
```

在具体的环境中要根据实际情况配置，不要照抄。

57.4.2 优化系统资源使用

在 Linux 中，资源的使用限制是由 limits.conf 来控制的，在文件中对于条目格式的设置有详细说明，因此在这里不多啰嗦了，直接添加以下条目：

```
[root@s8 bin]# cat /etc/security/limits.conf

…

* soft nofile  65535
* hard nofile  65535
```

另外，需要注意，要根据机器本身的性能来设置这两个值；还要注意，如果本机上的用户较多，那么最好是根据用户设置它们的值，而不要使用“*”。

另外一种设置资源使用的方法就是通过 Varnish 的启动脚本来设置，在后面有说明。

57.5 使用 Varnish

在前面我们已经将 Varnish 安装完成，下面我们来看看它的用法。在 Varnish 中有一个概念，那就是“backend”或“origin”服务器，后台服务器为 Varnish 提供内容，而 Varnish 正是将这些内容加速了访问。在此基础上就不难明白了，我们的第一个任务就是告诉 Varnish 去哪里找内容，换句话说就是指明后台服务器。使用你喜欢的文本编辑器编辑 Varnish 的默认配置文件，该文件的位置根据不同的安装方式可能会有两种位置，如果使用 rpm 包安装，那么会在 /etc/varnish/default.vcl 位置；如果使用源代码编译安装，那么将会由在安装时使用的选项 --prefix 决定。

例如，在这里我们的配置文件位于：

```
/usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl
```

在默认的配置文件中，所有的配置语句都被注释掉了，我们看到在最开始的某一部分，类似于以下内容：

```
# backend default {
# .host = "127.0.0.1";
# .port = "8080";
# }
```

我们可以将其注释去掉，然后根据实际情况修改。例如，将 8080 改为 80，将 127.0.0.1 改为 192.168.9.16，类似于这样：

```
backend default {
  .host = "192.168.9.16";
  .port = "80";
}
```

有了这一段代码之后，我们就定义了一台后台服务器，在 Varnish 中将会被称作 default，当 Varnish 需要获取内容时将会连接这台后台机器的 80 端口，以便获取想要的内容。Varnish 能够定义多个后台服务器，也可以将几个后台服务器做成后台集群，这样做的目的在于负载均衡。现在我们已经具备了基本的 Varnish 配置，可以启动 Varnish 进行基本测试了，我们将 Varnish 的监听端口定为 8080。

如果你将 varnish 命令添加到了 path 中，那么你可以直接在命令行运行 varnish 命令，如果没有将其添加到 path 中，那么就必须使用全路径方式（事实上我一直是这么做的，这种方式有它自己的好处）。但是在下面的使用中我们假设已经添加到 path 中（为的是将来在排版时方便，转行会少些）。

1. 执行权限

运行 Varnish 需要 root 权限，因此变成 root 权限执行以下命令：

```
[root@cache ~] pkill varnishd
```

之所以执行这条命令是确保没有其他的 Varnish 在运行。下面的命令用于启动运行 Varnish 服务器：

```
[root@cache ~] varnishd -f /usr/local/varnish-3.0.0-beta1/etc/varnish
/default.vcl -s malloc,1G -T 127.0.0.1:2000 -a 0.0.0.0:8080
```

2. varnishd 命令

看得出，我们在 varnish 命令行中添加了一些其他的选项。下面了解一下 varnish 命令的选项：

```
[root@cache sbin]# ./varnishd --help
./varnishd: invalid option -- -
usage: varnishd [options]
-a address:port # HTTP 监听的 IP 地址和端口号
-b address:port # 后台服务器的 IP 地址和端口号
# -b <主机名或者 IP 地址>
# -b '<主机名或者 IP 地址>:<服务端口>'
-C  # 将 VCL 代码编译为 C 语言
-d  # 调试模式
-f file # 使用指定的 VCL 脚本作为配置文件
-F       # 运行在前台
-h kind[,hashoptions]   # 指定哈希算法
#   -h critbit [default]
#   -h simple list
#   -h classic
#   -h classic,<buckets>
-i identity       # 表示 varnish 实例的身份
-l shl,free,fill  #共享内存文件的大小
#   shl: space for SHL records [80m]
#   free: space for other allocations [1m]
#   fill: prefill new file [+]
-M address:port #使用 CLI 连接 master 进程
-n dir  # 指定 varnishd 工作的目录
-P file # 指定 PID file
-p param=value             # 设置参数
-s kind[,storageoptions]  # 设定后台存储
#   -s malloc
#   -s file  [default: use /tmp]
#   -s file,<dir or file>
#   -s file,<dir or file>,<size>
#   -s persist{experimenta}
#   -s file,<dir or file>,<size>,<granularity>
-t               # 默认 TTL
-S secret-file  # 用于 CLI 密钥验证
-T address:port # Telnet 监听的 IP 地址和端口号
-V                    # 显示版本号
-w int[,int[,int]]  # 设定线程数量
```

```
    #   -w <fixed_count>
    #   -w min,max
    # -w min,max,timeout [default: -w2,500,300]
  -u user   # 指定启动后降级运行的用户
```

因此，我们上面指定的参数将是以下的解释：

```
-f /usr/local/etc/varnish/default.vcl
```

使用-f 选项指定出 Varnish 所使用的配置文件为/usr/local/etc/varnish/default.vcl。

```
-s malloc,1G
```

该选项用于选择 Varnish 时服务所使用的存储类型，Varnish 将会使用该容器存放从后端服务器获取的内容。在这里我们使用了 malloc，这种方式是使用内存存储内容的。注意一下书写格式，在逗号之后定义了所使用内存的大小，在这里我们定义了 1GB。

```
-T 127.0.0.1:2000
```

Varnish 内置了一个基于文本界面的管理接口，激活该接口能够在 Varnish 服务器不停止的前提下对其进行管理。我们可以指定这个端口，以避免与现有端口发生冲突。另外，要确保不能向外界暴露管理接口，这涉及到安全的问题。通过 Varnish 管理接口你可以很容易地获取对根文件系统的访问权限。因此，推荐使用监听的地址为 127.0.0.1，而不要完全相信防火墙规则来约束访问。

```
-a 0.0.0.0:8080
```

指定了 Varnish 服务器监听的端口号为 8080，由此端口来接收进入的 HTTP 请求，在生产环境中我们可能指定的端口是 80，这也是 Varnish 的默认值。

现在我们已经成功地运行了 Varnish 服务器。为了验证一下它是否能正确地工作，我们可以访问一下 http://192.168.9.2:8080/，看看是否访问的是 192.168.9.16 服务器上 80 端口的服务器。

3. varnishlog 命令

在 Varnish 中，一个很有趣的功能是日志如何工作，Varnish 替代了通用的日志记录文件，而且使用了共享内存段。当内存段被用到末尾时，记录将会重新开始，将旧的数据覆盖掉。与传统的方法相比（记录在磁盘文件中），这个功能大大地提高了处理速度，而且节省了磁盘空间。

命令 varnishlog 是 Varnish 提供的工具之一，我们可以通过它来查看 Varnish 记录的日志。varnishlog 给出的是原始日志，通过它可以看到记录到日志中的所有记录。例如，我们在命令行中执行该命令：

```
[root@cache bin]# ./varnishlog
0 CLI  - Rd ping
0 CLI  - Wr 200 19 PONG 1305880226 1.0
0 CLI  - Rd ping
0 CLI  - Wr 200 19 PONG 1305880229 1.0
0 CLI  - Rd ping
0 CLI  - Wr 200 19 PONG 1305880232 1.0
^C
```

这些日志是 Varnish 的 master 进程在检测 cache 进程，以便确定一切正常。现在我们回到浏览器再重新载入前面访问过的网页，同时监控日志：

```
[root@cache bin]# ./varnishlog
   12 BackendCloce - default
   12 BackendOpen  b default 192.168.9.2 44398 192.168.3.194 80
   12 TxRequestb GET
   12 TxURLb /
   12 TxProtocol   b HTTP/1.1
   12 TxHeader b Host: 192.168.9.2:8080
   12 TxHeader b User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1
   12 TxHeader b Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
   12 TxHeader b Accept-Language: zh,en-us;q=0.7,en;q=0.3
   12 TxHeader b Accept-Encoding: gzip,deflate
   12 TxHeader b Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
   12 TxHeader b Cookie:  PCSFirstTime=110-12-16-8-22-3;
_PCSReturnTime=110-12-16-8-51-58; _PCSReturnCount=2; AJSTAT_ok_times=1;
rtime=0; ltime=1292460724834; cnzz_eid=50889502-1292454240-;
__FTcilgffgh=2010-12-16-8-22-16; __NRUcilgffgh=1292458936277;
  RTcilgffgh=2010-12
   12 TxHeader b If-Modified-Since: Tue, 17 May 2011 01:39:01 GMT
   12 TxHeader b X-Forwarded-For: 192.168.9.128
   12 TxHeader b X-Varnish: 1894359135
   12 RxProtocol   b HTTP/1.1
   12 RxStatus b 304
   12 RxResponse   b Not Modified
   12 RxHeader b Server: nginx/0.8.54
   12 RxHeader b Date: Fri, 20 May 2011 07:44:59 GMT
   12 RxHeader b Last-Modified: Tue, 17 May 2011 01:39:01 GMT
   12 RxHeader b Connection: keep-alive
   12 RxHeader b Expires: Fri, 20 May 2011 08:44:59 GMT
   12 RxHeader b Cache-Control: max-age=3600
   12 Fetch_Body   b 0 0 0
   12 Length   b 0
   12 BackendReuse b default
   10 SessionOpen  c 192.168.9.128 2671 0.0.0.0:8080
   10 ReqStart c 192.168.9.128 2671 1894359135
   10 RxRequestc GET
   10 RxURLc /
   10 RxProtocol   c HTTP/1.1
   10 RxHeader c Host: 192.168.9.2:8080
   10 RxHeader c User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1
   10 RxHeader c Accept: text/html,application/xhtml+xml,
```

```
application/xml;q=0.9,*/*;q=0.8
    10 RxHeader c Accept-Language: zh,en-us;q=0.7,en;q=0.3
    10 RxHeader c Accept-Encoding: gzip,deflate
    10 RxHeader c Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
    10 RxHeader c Keep-Alive: 115
    10 RxHeader c Connection: keep-alive
    10 RxHeader c Cookie:  PCSFirstTime=110-12-16-8-22-3;
 _PCSReturnTime=110-12-16-8-51-58; _PCSReturnCount=2; AJSTAT_ok_times=1;
 rtime=0; ltime=1292460724834; cnzz_eid=50889502-1292454240-;
  __FTcilgffgh=2010-12-16-8-22-16; __NRUcilgffgh=1292458936277;
  __RTcilgffgh=2010-12
    10 RxHeader c If-Modified-Since: Tue, 17 May 2011 01:39:01 GMT
    10 VCL call c recv pass
    10 VCL call c hash
    10 Hash c /
    10 Hash c 192.168.9.2:8080
    10 VCL_return   c hash
    10 VCL_call c pass pass
    10 Backend  c 12 default default
    10 TTL  c 1894359135 RFC 120 1305880815 0 0 0 0
    10 VCL call c fetch hit for pass
    10 ObjProtocol  c HTTP/1.1
    10 ObjResponse  c Not Modified
    10 ObjHeaderc Server: nginx/0.8.54
    10 ObjHeaderc Date: Fri, 20 May 2011 07:44:59 GMT
    10 ObjHeaderc Last-Modified: Tue, 17 May 2011 01:39:01 GMT
    10 ObjHeaderc Expires: Fri, 20 May 2011 08:44:59 GMT
    10 ObjHeaderc Cache-Control: max-age=3600
    10 VCL call c deliver deliver
    10 TxProtocol   c HTTP/1.1
    10 TxStatus c 304
    10 TxResponse   c Not Modified
    10 TxHeader c Server: nginx/0.8.54
    10 TxHeader c Last-Modified: Tue, 17 May 2011 01:39:01 GMT
    10 TxHeader c Expires: Fri, 20 May 2011 08:44:59 GMT
    10 TxHeader c Cache-Control: max-age=3600
    10 TxHeader c Accept-Ranges: bytes
    10 TxHeader c Date: Fri, 20 May 2011 08:40:15 GMT
    10 TxHeader c X-Varnish: 1894359135
    10 TxHeader c Age: 0
    10 TxHeader c Via: 1.1 varnish
    10 TxHeader c Connection: keep-alive
    10 Length   c 0
```

```
    10 ReqEnd   c 1894359135 1305880815.435889482 1305880815.437436104
0.000055790 0.001511574 0.000035048
```

- 第一列是一个随意的数值，它只是确定了一个请求。
- 第二列是日志消息的 tag，所有的日志条目都被打上了 tag，它简短地指明了日志的类型，由 Rx 开始的表示这是 Varnish 收到的数据，而 Tx 则表示是发送的数据。
- 第三列告诉我们这个数据是来源于客户端还是发送到客户端，以及从后台发送的数据或者是后台返回的数据，在这里注意这一列的标志 c，即表示客户端；b 则表示后台。
- 第四列是被记录的数据。

看下面的选项，varnishlog 命令可以使用这些选项进行相当多的过滤。以下是基本选项：

-b	仅显示 Varnish 和后台服务器之间的流量，这个参数在进行优化缓存命中率时非常有用
-c	功能同上，但是它显示的是客户端的流量
-i tag	仅显示某一指定 tag 的行，例如，"，"varnishlog -i SessionOpen"将显示新的会话打开。需要注意的一点是，tag 对大小写不敏感，就是说不区分大小写
-I Regex	通过正则表达式来过滤记录，仅显示匹配行。例如，"varnishlog -c -i RxHeader -I Cookie"则表示显示所有来自于客户端的 cookie headers
-O	不通过请求 ID 对日志分组记录

例如：

```
 [root@cache bin]# ./varnishlog -b
    12 BackendOpen  b default 192.168.3.139 36904 192.168.3.194 80
    12 TxRequestb GET
    12 TxURLb /
    12 TxProtocol   b HTTP/1.1
    12 TxHeader b Host: 192.168.3.139:8080
    12 TxHeader b User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1
    12 TxHeader b Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
    12 TxHeader b Accept-Language: zh,en-us;q=0.7,en;q=0.3
    12 TxHeader b Accept-Encoding: gzip,deflate
    12 TxHeader b Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
    12 TxHeader b Cookie:  PCSFirstTime=110-12-16-8-22-3;
 PCSReturnTime=110-12-16-8-51-58;  PCSReturnCount=2;  AJSTAT ok times=1;
rtime=0; ltime=1292460724834; cnzz_eid=50889502-1292454240-;
__FTcilgffgh=2010-12-16-8-22-16; __NRUcilgffgh=1292458936277;
__RTcilgffgh=2010-12
    12 TxHeader b If-Modified-Since: Tue, 17 May 2011 01:39:01 GMT
    12 TxHeader b X-Forwarded-For: 192.168.3.248
    12 TxHeader b X-Varnish: 1894359136
    12 RxProtocol   b HTTP/1.1
    12 RxStatus b 304
    12 RxResponse   b Not Modified
```

```
12 RxHeader b Server: nginx/0.8.54
12 RxHeader b Date: Fri, 20 May 2011 23:07:38 GMT
12 RxHeader b Last-Modified: Tue, 17 May 2011 01:39:01 GMT
12 RxHeader b Connection: keep-alive
12 RxHeader b Expires: Sat, 21 May 2011 00:07:38 GMT
12 RxHeader b Cache-Control: max-age=3600
12 Fetch Body   b 0 0 0
12 Length   b 0
12 BackendReuse b default
```

57.6 缓存大小的设置

选择多少内存给 Varnish 使用是一个棘手的问题，下面是一些可以参考的方面。

varnishstat 命令

多大的热区数据集（hot data set），对于门户或新闻网站，与主页相关的组成部分，例如，图片，css 等这些对象，由于现在的内存已经不是特别昂贵，因此对于缓存每一个对象的成本也就不是很高了，但是仍然要调整缓存的内容，可以通过 varnishstat 命令或者其他工具来观察，例如：

```
[root@s8 bin]# ./varnishstat

0+05:11:27 s8.xx.cn
Hitrate ratio:   10   64   64
Hitrate avg: 0.9714   0.9736   0.9736

 2468338   264.72   132.09 Client connections accepted
 2468188   264.72   132.08 Client requests received
 2311199   254.73   123.68 Cache hits
  156897 9.99 8.40 Cache misses
2719 0.00 0.15 Backend conn. success
  154271 9.99 8.26 Backend conn. reuses
1629 0.00 0.09 Backend conn. was closed
  155902 9.99 8.34 Backend conn. recycles
   40767 2.00 2.18 Fetch with Length
  116130 7.99 6.21 Fetch chunked
   1 0.00 0.00 Fetch failed
 100  ..   N struct sess_mem
  87  ..   N struct sess
1017  ..   N struct object
1027  ..   N struct objectcore
2745  ..   N struct objecthead
   3  ..   N struct vbe_conn
  10  ..   N worker threads
```

```
 27 0.00 0.00 N worker threads created
169 0.00 0.01 N overflowed work requests
  1  ..   N backends
 155458  ..   N expired objects
 629549  ..   N LRU moved objects
1803314   172.8296.50 Objects sent with write
2468337   264.72   132.09 Total Sessions
2468188   264.72   132.08 Total Requests
 92 0.00 0.00 Total pipe
 156896 9.99 8.40 Total fetch
  594577993 60518.43 31817.73 Total header bytes
39618375195   4665995.81   2120103.56 Total body bytes
2468337   264.72   132.09 Session Closed
98775114 10405.90  5285.77 SHM records
10219791  1079.85   546.89 SHM writes
 31 0.00 0.00 SHM flushes due to overflow
  18396 1.00 0.98 SHM MTX contention
 38 0.00 0.00 SHM cycles through buffer
 31581917.9816.90 SMA allocator requests
2051  ..   SMA outstanding allocations
42277723  ..   SMA outstanding bytes
16580272294  ..   SMA bytes allocated
16537994571  ..   SMA bytes free
  1 0.00 0.00 SMS allocator requests
419  ..   SMS bytes allocated
419  ..   SMS bytes freed
 156898 9.99 8.40 Backend requests made
  1 0.00 0.00 N vcl total
  1 0.00 0.00 N vcl available
  1  ..   N total active purges
  1 0.00 0.00 N new purges added
2468030   264.72   132.07 HCB Lookups without lock
 125200 4.00 6.70 HCB Lookups with lock
 125200 4.00 6.70 HCB Inserts
```

通过 varnishstat 或者其他工具观察 n_lru_nuked 的统计。例如：

```
[root@cache bin]# ./varnishstat -f n lru nuk

0+00:01:57
Hitrate ratio:444
Hitrate avg: 0.0132   0.0132   0.0132
```

如果有大量的 LRU 活动对象，那么由于缓存空间的约束将会把一些对象驱赶出去，在这种情况下，你需要增加缓存的大小。

57.7 VCL 配置

Varnish 有一个巨大的配置系统，许多其他系统使用的配置指令基本上是通过切换打开或者是关闭指令，就像开关一样，要么是打开状态，要么就是关闭状态。每一种服务器都有自己的配置语言指令，例如，Nginx 有它自己的配置指令，而 Squid 也有自己的配置指令，因此 Varnish 也就不例外了，它使用了一个领域特定语言，被称为——Varnish 配置语言（Varnish Configuration Language），缩写为 VCL。在 Varnish 启动时，VCL 会将配置文件转换为二进制代码，并且在被访问时执行。

VCL 文件被分割为许多子程序，不同的子程序在不同的时间执行。一种是当我们执行请求时执行，另一种则是从后端服务器获取文件时执行。

Varnish 将会在它工作的不同阶段执行这些代码，因为这些代码是一行接着一行执行的，因此优先权不是问题。在某些时候，在子程序中调用了一个 action，然后执行子程序停止。

如果在执行子程序时没有调用 action，那么在执行到代码末尾时，Varnish 将会调用一些内建的 VCL 代码。这些可以查看默认 default.vcl 中的注解。

在我们对配置文件的修改中，99%的会对 vcl_recv 和 vcl_fetch 这两种子程序进行修改。下面看一个使用中的 VCL 配置文件：

```
[root@cache varnish]# pwd
/usr/local/varnish-3.0.0-beta1/etc/varnish
[root@cache varnish]# vi default.vcl

 backend ccm {
   .host = "192.168.3.155";
   .port = "80";

 }

 sub vcl_recv {

   set req.backend = ccm;
   if  (req.request != "GET" && req.request != "HEAD")  {
   return (pipe) ;
}
else {
   return (lookup) ;
 }

 }

 sub vcl pipe {
 return  (pipe) ;
 }
```

```
sub vcl_pass {
return (pass);
}

sub vcl_hit {
return (deliver);
}

sub vcl_miss {
return (fetch);
}


sub vcl_fetch {

return (deliver);
}


sub vcl_deliver {
return (deliver);
}


sub vcl_fetch {
  if (req.request == "GET" && req.url ~ "\.(xml|css|txt|js|JPG|jpg|gif|
swf|GIF|zip|shtml|html|htm)$") {
   set beresp.ttl = 500s;
}
  else if (req.request == "GET" && req.url ~ "index") {
   set beresp.ttl = 180s;
}
  else if (req.request == "GET" && req.url ~ "/$") {
   set beresp.ttl = 180s;
}
  else {
   set beresp.ttl = 200s;
}
}
```

57.8 Varnish 的启动与停止

在分析源代码目录时我们注意到这样一个目录结构：

```
[root@cache redhat]# tree
.
|-- Makefile.am
|-- Makefile.in
|-- README.redhat
|-- TODO
|-- varnish.initrc
|-- varnish.logrotate
|-- varnish.spec
|-- varnish.sysconfig
|-- varnish_reload_vcl
|-- varnishlog.initrc
'-- varnishncsa.initrc

0 directories, 11 files
```

在这个目录中除了我们下面要讨论的两个文件外，还有几个和日志相关的文件。我们将在下面进行介绍。

1. varnish.initrc 文件

从 varnish.initrc 文件的名字可以看出来它是用于 init 启动的。下面是其内容：

```
[root@cache redhat]# more varnish.initrc
#! /bin/sh
#
# varnish Control the Varnish Cache
#
# chkconfig: - 90 10
# description: Varnish is a high-perfomance HTTP accelerator
# processname: varnishd
# config: /etc/sysconfig/varnish
# pidfile: /var/run/varnishd.pid

### BEGIN INIT INFO
# Provides: varnish
# Required-Start: $network $local_fs $remote_fs
# Required-Stop: $network $local_fs $remote_fs
# Default-Start:
# Default-Stop:
# Should-Start: $syslog
# Short-Description: start and stop varnishd
```

```
# Description: Varnish is a high-perfomance HTTP accelerator
### END INIT INFO

# Source function library.
. /etc/init.d/functions

retval=0 //设定运行参数
pidfile=/var/run/varnish.pid

exec="/usr/sbin/varnishd"
reload_exec="/usr/bin/varnish_reload_vcl"
prog="varnishd"
config="/etc/sysconfig/varnish"
lockfile="/var/lock/subsys/varnish"

# Include varnish defaults
[ -e /etc/sysconfig/varnish ] && . /etc/sysconfig/varnish


start () {

if [ ! -x $exec ]
then
echo $exec not found
exit 5
fi

if [ ! -f $config ]
then
echo $config not found
exit 6
fi
echo -n "Starting Varnish Cache: "

# Open files (usually 1024, which is way too small for varnish)
ulimit -n ${NFILES:-131072} //设定资源使用限制

# Varnish wants to lock shared memory log in memory.
ulimit -l ${MEMLOCK:-82000} //设定日志共享内存

# $DAEMON_OPTS is set in /etc/sysconfig/varnish. At least, one
# has to set up a backend, or /tmp will be used, which is a bad idea.
if [ "$DAEMON_OPTS" = "" ]; then
```

```
    echo "\$DAEMON_OPTS empty."
    echo -n "Please put configuration options in $config"
    return 6
    else
      # Varnish always gives output on STDOUT
      daemon --pidfile $pidfile  $exec -P $pidfile "$DAEMON OPTS" > /dev/null
2>&1
      retval=$?
      if [ $retval -eq 0 ]
      then
      touch $lockfile
      echo success
      echo
      else
      echo failure
      echo
      fi
      return $retval
    fi
    }

    stop () {
    echo -n "Stopping Varnish Cache: "
    killproc -p $pidfile $prog
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
    }

    restart () {
    stop
    start
    }

    reload () {
    if [ "$RELOAD VCL" = "1" ]
    then
    $reload_exec
    else
    force_reload
    fi
    }
```

```
force_reload () {
restart
}

rh_status () {
status -p $pidfile $prog
}

rh_status_q () {
rh_status >/dev/null 2>&1
}

configtest () {
 if [ -f "$VARNISH VCL CONF" ]; then
 $exec -f "$VARNISH_VCL_CONF" -C -n /tmp > /dev/null && echo "Syntax ok"
  else
echo "VARNISH_VCL_CONF is  unset or does not point to a file"
 fi
}

# See how we were called.
case "$1" in
start)
rh_status_q && exit 0
$1
;;
stop)
rh status q || exit 0
$1
;;
restart)
$1
;;
reload)
rh status q || exit 7
$1
;;
force-reload)
force_reload
;;
status)
rh_status
```

```
;;
condrestart|try-restart)
rh_status_q || exit 0
restart
;;
configtest)
configtest
;;
*)
echo "Usage: $0 {start|stop|status|restart|condrestart|try-restart|
reload|force-reload}"

exit 2
esac

exit $?

[root@cache redhat]#
```

分析该启动文件需要的命令以及参数配置和参数配置文件，然后对其进行适当的修改，以便我们在安装 Varnish 服务器时的定制。

2. varnish.sysconfig 文件

从文件名可以看出，varnish.sysconfig 用于设定启动配置参数：

```
[root@cache redhat]# cat  varnish.sysconfig
# Configuration file for varnish
#
# /etc/init.d/varnish expects the variable $DAEMON_OPTS to be set from this
# shell script fragment.
#

# Maximum number of open files (for ulimit -n)
NFILES=131072

# Locked shared memory (for ulimit -l)
# Default log size is 82MB + header
MEMLOCK=82000

# Maximum size of corefile (for ulimit -c). Default in Fedora is 0
# DAEMON_COREFILE_LIMIT="unlimited"

# Set this to 1 to make init script reload try to switch vcl without restart.
# To make this work, you need to set the following variables
# explicit: VARNISH_VCL_CONF, VARNISH_ADMIN_LISTEN_ADDRESS,
```

```
# VARNISH_ADMIN_LISTEN_PORT, VARNISH_SECRET_FILE, or in short,
# use Alternative 3, Advanced configuration, below
RELOAD_VCL=1

# This file contains 4 alternatives, please use only one.

## Alternative 1, Minimal configuration, no VCL
#
# Listen on port 6081, administration on localhost:6082, and forward to
# content server on localhost:8080.  Use a fixed-size cache file.
#
#DAEMON OPTS="-a :6081 \
# -T localhost:6082 \
# -b localhost:8080 \
# -u varnish -g varnish \
# -s file,/var/lib/varnish/varnish_storage.bin,1G"


## Alternative 2, Configuration with VCL
#
# Listen on port 6081, administration on localhost:6082, and forward to
# one content server selected by the vcl file, based on the request.  Use a
# fixed-size cache file.
#
#DAEMON_OPTS="-a :6081 \
# -T localhost:6082 \
# -f /etc/varnish/default.vcl \
# -u varnish -g varnish \
# -S /etc/varnish/secret \
# -s file,/var/lib/varnish/varnish storage.bin,1G"


## Alternative 3, Advanced configuration
#
# See varnishd (1)  for more information.
#
# # Main configuration file. You probably want to change it :)
VARNISH_VCL_CONF=/etc/varnish/default.vcl
#
# # Default address and port to bind to
# # Blank address means all IPv4 and IPv6 interfaces, otherwise specify
# # a host name, an IPv4 dotted quad, or an IPv6 address in brackets.
# VARNISH_LISTEN_ADDRESS=
```

```
VARNISH_LISTEN_PORT=6081
#
# # Telnet admin interface listen address and port
VARNISH ADMIN LISTEN ADDRESS=127.0.0.1
VARNISH ADMIN LISTEN PORT=6082
#
# # Shared secret file for admin interface
VARNISH_SECRET_FILE=/etc/varnish/secret
#
# # The minimum number of worker threads to start
VARNISH_MIN_THREADS=1
#
# # The Maximum number of worker threads to start
VARNISH MAX THREADS=1000
#
# # Idle timeout for worker threads
VARNISH_THREAD_TIMEOUT=120
#
# # Cache file location
VARNISH STORAGE FILE=/var/lib/varnish/varnish storage.bin
#
# # Cache file size: in bytes, optionally using k / M / G / T suffix,
# # or in percentage of available disk space using the % suffix.
VARNISH_STORAGE_SIZE=1G
#
# # Backend storage specification
VARNISH STORAGE="file,${VARNISH STORAGE FILE},${VARNISH STORAGE SIZE}"
#
# # Default TTL used when the backend does not specify one
VARNISH TTL=120
#
# # DAEMON_OPTS is used by the init script.  If you add or remove options,
make
# # sure you update this section, too.
DAEMON OPTS="-a ${VARNISH LISTEN ADDRESS}:${VARNISH LISTEN PORT} \
 -f ${VARNISH VCL CONF} \
 -T ${VARNISH ADMIN LISTEN ADDRESS}:${VARNISH ADMIN LISTEN PORT} \
 -t ${VARNISH_TTL} \
 -w ${VARNISH_MIN_THREADS},${VARNISH_MAX_THREADS},${VARNISH_THREAD
_TIMEOUT} \
 -u varnish -g varnish \
 -S ${VARNISH SECRET FILE} \
 -s ${VARNISH_STORAGE}"
```

```
#


## Alternative 4, Do It Yourself. See varnishd (1)  for more information.
#
```

在这个文件中给出了 4 种设置 DAEMON_OPTS 即守护进程变量的参数，我们根据实际情况选择一种即可。

3. 应用配置

要使用 varnish.initrc 和 varnish.sysconfig 这两个文件还需要对其进行修改。首先我们做以下操作，以便将其放在合适的位置上，然后对其进行修改：

```
[root@cache redhat]#useradd varnish   //首先添加 varnishd 用户
[root@cache redhat]# cp varnish.initrc /etc/init.d/
[root@cache redhat]# cd /etc/init.d/
[root@cache init.d]# mv varnish.initrc varnishd
[root@cache redhat]# cp varnish reload vcl  /usr/local/varnish-3.0.0
-beta1/bin/
[root@cache redhat]# cp varnish.sysconfig  /usr/local/varnish-3.0.0
-beta1/etc
```

根据实际情况将/usr/local/varnish-3.0.0-beta1/etc/varnish.sysconfig 变量配置文件修改为如下内容：

```
[root@cache init.d]# cat /usr/local/varnish-3.0.0-beta1/etc/varnish.
sysconfig
NFILES=131072

MEMLOCK=82000

RELOAD_VCL=1

VARNISH VCL CONF=/usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl
VARNISH LISTEN PORT=80
VARNISH ADMIN LISTEN ADDRESS=127.0.0.1
VARNISH ADMIN LISTEN PORT=6082
VARNISH_SECRET_FILE=/root/pass.file
VARNISH_MIN_THREADS=1
VARNISH_MAX_THREADS=1000
VARNISH_THREAD_TIMEOUT=120
VARNISH STORAGE SIZE=1G
VARNISH STORAGE="malloc,${VARNISH STORAGE SIZE}"
VARNISH TTL=120

DAEMON_OPTS="-a ${VARNISH_LISTEN_ADDRESS}:${VARNISH_LISTEN_PORT} \
  -f ${VARNISH_VCL_CONF} \
```

```
    -T ${VARNISH_ADMIN_LISTEN_ADDRESS}:${VARNISH_ADMIN_LISTEN_PORT} \
    -t ${VARNISH_TTL} \
    -w ${VARNISH_MIN_THREADS},${VARNISH_MAX_THREADS},${VARNISH_THREAD_
       TIMEOUT} \
    -u varnish -g varnish \
    -S ${VARNISH_SECRET_FILE} \
    -s ${VARNISH_STORAGE}"
```

然后对/etc/init.d/varnishd 文件做了以下修改：

```
[root@cache init.d]# cat varnishd
#! /bin/sh
#
# varnish Control the Varnish Cache
#
# chkconfig: - 90 10
# description: Varnish is a high-perfomance HTTP accelerator
# processname: varnishd
# config: /usr/local/varnish-3.0.0-beta1/etc/varnish.sysconfig
# pidfile: /var/run/varnishd.pid

### BEGIN INIT INFO
# Provides: varnish
# Required-Start: $network $local_fs $remote_fs
# Required-Stop: $network $local_fs $remote_fs
# Default-Start:
# Default-Stop:
# Should-Start: $syslog
# Short-Description: start and stop varnishd
# Description: Varnish is a high-perfomance HTTP accelerator
### END INIT INFO

# Source function library.
. /etc/init.d/functions

retval=0
pidfile=/var/run/varnish.pid

exec="/usr/local/varnish-3.0.0-beta1/sbin/varnishd"
reload_exec="/usr/local/varnish-3.0.0-beta1/bin/varnish_reload_vcl"
prog="varnishd"
config="/usr/local/varnish-3.0.0-beta1/etc/varnish.sysconfig"
lockfile="/var/lock/subsys/varnish"

# Include varnish defaults
```

```
[ -e /usr/local/varnish-3.0.0-beta1/etc/varnish.sysconfig ] && . /usr/local/varnish-3.0.0-beta1/etc/varnish.sysconfig

… //以下内容未做修改
```

添加 Varnish 服务器并设置启动级别：

```
[root@cache init.d]# chkconfig --add varnishd
[root@cache init.d]# chkconfig --levels 35 varnishd on
```

启动、停止 Varnish 服务器：

```
[root@cache init.d]# service varnishd start
Starting Varnish Cache:[  OK  ]
[root@cache init.d]#
[root@cache init.d]# ps -ef|grep varnishd
root  2602 1  0 10:20 ?00:00:00 /usr/local/varnish-3.0.0-beta1/sbin/varnishd -P /var/run/varnish.pid -a :80 -f /usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -T 127.0.0.1:6082 -t 120 -w 1,1000,120 -u varnish -g varnish -S /root/pass.file -s malloc,1G
varnish  2604  2602  0 10:20 ?00:00:00 /usr/local/varnish-3.0.0-beta1/sbin/varnishd -P /var/run/varnish.pid -a :80 -f /usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -T 127.0.0.1:6082 -t 120 -w 1,1000,120 -u varnish -g varnish -S /root/pass.file -s malloc,1G
[root@cache init.d]#
[root@cache init.d]# service varnishd stop
Stopping Varnish Cache:  [  OK  ]
```

57.9 Varnish 的访问日志

Varnish 的日志有两种，一种是原始日志，而另一种则是演变成 Apache 日志格式的日志。

这个功能根据实际情况选择，它会消耗较多的资源，另外也没必要每时每刻地去收集处理日志，可以根据时段来查看访问情况，一般我们都是通过日志来分析访问资源情况，以便为服务器优化提供确切的资料。

前面说了，在源代码安装包中有一个 redhat 的目录，它下面有以下文件被用于日志处理 varnishncsa.initrc 、varnish.logrotate 和 varnishlog.initrc。

1. varnishncsa.initrc 文件

从文件的名字来看就知道 varnishncsa.initrc 文件是用于初始化守护进程的，看一下其内容：

```
[root@cache redhat]# more varnishncsa.initrc
#! /bin/sh
#
# varnishncsa Control the Varnish NSCA logging daemon
#
```

```
# chkconfig: - 90 10
# description: Varnish Cache logging daemon
# processname: varnishncsa
# config:
# pidfile: /var/run/varnishncsa.pid

### BEGIN INIT INFO
# Provides: varnishncsa
# Required-Start: $network $local_fs $remote_fs
# Required-Stop: $network $local_fs $remote_fs
# Default-Start:
# Default-Stop:
# Short-Description: start and stop varnishncsa
# Description: Varnish Cache NSCA logging daemon
### END INIT INFO

# Source function library.
. /etc/init.d/functions

retval=0
pidfile="/var/run/varnishncsa.pid"
lockfile="/var/lock/subsys/varnishncsa"
logfile="/var/log/varnish/varnishncsa.log"

exec="/usr/bin/varnishncsa"
prog="varnishncsa"

DAEMON OPTS="-a -w $logfile -D -P $pidfile"

# Include varnishncsa defaults
[ -e /etc/sysconfig/varnishncsa ] && . /etc/sysconfig/varnishncsa

start () {

if [ ! -x $exec ]
then
echo $exec not found
exit 5
fi

echo -n "Starting varnish ncsa logging daemon: "

daemon --pidfile $pidfile $exec "$DAEMON_OPTS"
```

```
echo
return $retval
}

stop () {
echo -n "Stopping varnish ncsa logging daemon: "
killproc -p $pidfile $prog
retval=$?
echo
[ $retval -eq 0 ] && rm -f $lockfile
return $retval
}

restart () {
stop
start
}

reload () {
restart
}

force_reload () {
restart
}

rh status () {
status -p $pidfile $prog
}

rh_status_q () {
rh_status >/dev/null 2>&1
}

# See how we were called.
case "$1" in
start)
rh_status_q && exit 0
$1
;;
stop)
rh status q || exit 0
$1
```

```
;;
restart)
$1
;;
reload)
rh status q || exit 7
$1
;;
force-reload)
force_reload
;;
status)
rh status
;;
condrestart|try-restart)
rh_status_q || exit 0
restart
;;
*)
 echo "Usage: $0 {start|stop|status|restart|condrestart|try-restart|
reload|force-reload}"

exit 2
esac

exit $?

[root@cache redhat]#
```

首先将 varnishncsa.initrc 文件文件复制到/etc/init.d/目录下：

```
[root@cache redhat]# cp varnishncsa.initrc  /etc/init.d/varnishncsa
```

它的内容和 varnish.initrc 文件差不多，因此要想使用该文件，还需要根据实际情况进行修改：

```
exec="/usr/local/varnish-3.0.0-beta1/bin/varnishncsa"
prog="varnishncsa"

DAEMON_OPTS="-a -w $logfile -D -P $pidfile"

# Include varnishncsa defaults
[ -e /usr/local/varnish-3.0.0-beta1/bin/varnishncsa ] && . /usr/local/
varnish-3.0.0-beta1/bin/varnishncsa
```

添加服务并设置启动级别：

```
[root@cache init.d]# chkconfig --add  varnishncsa
[root@cache init.d]# chkconfig --levels  35  varnishncsa on
```

创建存放日志的目录：

```
[root@cache init.d]#mkdir /var/log/varnish/
```

启动日志守护进程：

```
[root@cache init.d]# service varnishncsa start
Starting varnish ncsa logging daemon:  [  OK  ]
```

2. varnish.logrotate 文件

由于访问日志的不断增大，因此在分析和查看时会带来不便，因此有必要对日志进行切割，下面我们看一下 varnish.logrotate 文件：

```
[root@cache redhat]# cat varnish.logrotate
/var/log/varnish/*.log {
missingok
notifempty
sharedscripts
delaycompress
postrotate
/bin/kill -HUP 'cat /var/run/varnishlog.pid 2>/dev/null' 2> /dev/null ||
true
/bin/kill -HUP 'cat /var/run/varnishncsa.pid 2>/dev/null' 2> /dev/null ||
true
endscript
}
[root@cache redhat]#
```

也可对该文件做适当的修改，然后放在/etc/logrotate.d 目录下。在我使用的环境中没有这么做，包括 varnishncsa、varnishlog 作为服务开机启动，都没这么做。通常我使用这两个工具抓取日志进行检测，以便分析日志发现问题。

例如会出现类似以下的日志文件：

```
[root@cache varnish]# ls
varnish.log  varnish.log.1  varnish.log.2
varnishncsa.log  varnishncsa.log.1  varnishncsa.log.2
```

3. varnishlog.initrc 文件

看以下 varnishlog.initrc 文件的内容和上面的文件基本相似，因此需要修改的内容也不多：

```
[root@cache redhat]# cat varnishlog.initrc
#! /bin/sh
#
# varnishlog Control the Varnish logging daemon
#
# chkconfig: - 90 10
# description: Varnish Cache logging daemon
# processname: varnishlog
# config:
```

```
# pidfile: /var/run/varnishlog.pid

### BEGIN INIT INFO
# Provides: varnishlog
# Required-Start: $network $local fs $remote fs
# Required-Stop: $network $local fs $remote fs
# Default-Start:
# Default-Stop:
# Short-Description: start and stop varnishlog
# Description: Varnish Cache logging daemon
### END INIT INFO

# Source function library.
. /etc/init.d/functions

retval=0
pidfile="/var/run/varnishlog.pid"
lockfile="/var/lock/subsys/varnishlog"
logfile="/var/log/varnish/varnish.log"

exec="/usr/bin/varnishlog"
prog="varnishlog"

DAEMON_OPTS="-a -w $logfile -D -P $pidfile"

# Include varnishlog defaults
[ -e /etc/sysconfig/varnishlog ] && . /etc/sysconfig/varnishlog

start () {

if [ ! -x $exec ]
then
echo $exec not found
exit 5
fi

echo -n "Starting varnish logging daemon: "

daemon --pidfile $pidfile $exec "$DAEMON_OPTS"
echo
return $retval
}
```

```
stop () {
echo -n "Stopping varnish logging daemon: "
killproc -p $pidfile $prog
retval=$?
echo
[ $retval -eq 0 ] && rm -f $lockfile
return $retval
}

restart () {
stop
start
}

reload () {
restart
}

force reload () {
restart
}

rh_status () {
status -p $pidfile $prog
}

rh status q () {
rh status >/dev/null 2>&1
}

# See how we were called.
case "$1" in
start)
rh status q && exit 0
$1
;;
stop)
rh_status_q || exit 0
$1
;;
restart)
$1
;;
```

```
reload)
rh_status_q || exit 7
$1
;;
force-reload)
force reload
;;
status)
rh_status
;;
condrestart|try-restart)
rh status q || exit 0
restart
;;
*)
echo "Usage: $0 {start|stop|status|restart|condrestart|try-restart|
reload|force-reload}"

exit 2
esac

exit $?
```

首先将 varnishlog.initrc 文件复制到/etc/init.d/目录下：

```
[root@cache redhat]# cp varnishlog.initrc  /etc/init.d/varnishlog
```

修改以下两处即可：

```
exec="/usr/local/varnish-3.0.0-beta1/bin/varnishlog"

…

[-e/usr/local/varnish-3.0.0-beta1/bin/varnishlog]&&./usr/local/varnish-3
.0.0-beta1/bin/varnishlog
```

添加服务并设置启动级别：

```
[root@cache init.d]# chkconfig --add  varnishlog
[root@cache init.d]# chkconfig --levels  35  varnishlog on
```

创建存放日志的目录：

```
[root@cache init.d]#mkdir /var/log/varnish/
```

启动日志守护进程：

```
[root@cache init.d]# service varnishlog start
Starting varnish logging daemon:[  OK  ]
```

如果出现以下情况：

```
[root@cache init.d]# service varnishlog start
varnishlog: unrecognized service
```

这是因为没有执行权限所造成的，那么执行以下命令：

```
[root@cache init.d]# chmod 755 varnishlog
```

57.10 守护进程 varnishd

Varnishd 是 http 加速器守护进程。它接收客户端发送来的请求，将它们传递到后台服务器，并且将返回的文档缓存，以便以后的请求获取同样的文档时有 Varnish 的缓存直接提供。

1. 语法格式

```
varnishd [-a address[:port]] [-b host[:port]] [-d] [-F]
     [-f config] [-g group] [-h type[,options]]
     [-i identity] [-l shmlogsize] [-n name] [-P file]
     [-p param=value] [-s type[,options]]
     [-T address[:port]] [-t ttl] [-u user] [-V]
     [-w min[,max[,timeout]]]
```

有关参数可以参考“使用 Varnish”部分。另外，不同的版本有不同的参数，例如，以下是 2.1.5 的参数：

```
[root@s8 varnish-2.1.5]# /usr/local/varnish-2.1.5/sbin/varnishd --h
/usr/local/varnish-2.1.5/sbin/varnishd: invalid option -- -
usage: varnishd [options]
-a address:port  # HTTP listen address and port
-b address:port  # backend address and port
 #-b <hostname_or_IP>
 #-b '<hostname_or_IP>:<port_or_service>'
-C   # print VCL code compiled to C language
-d   # debug
-f file  # VCL script
-F   # Run in foreground
-h kind[,hashoptions]# Hash specification
 #   -h simple list
 #   -h classic  [default]
 #   -h classic,<buckets>
-i identity  # Identity of varnish instance
-l bytesize  # Size of shared memory log
-M address:port  # CLI-master to connect to.
-n dir   # varnishd working directory
-P file  # PID file
-p param=value   # set parameter
-s kind[,storageoptions] # Backend storage specification
 #   -s malloc
 #   -s file  [default: use /tmp]
 #   -s file,<dir_or_file>
```

```
    #   -s file,<dir_or_file>,<size>
    #   -s file,<dir_or_file>,<size>,<granularity>
   -t   # Default TTL
   -S secret-file   # Secret file for CLI authentication
   -T address:port  # Telnet listen address and port
   -V   # version
   -w int[,int[,int]]   # Number of worker threads
    #   -w <fixed_count>
    #   -w min,max
    #   -w min,max,timeout [default: -w2,500,300]
   -u user  # Priviledge separation user id
```

我们再看一下 3.0.0-beta1 的参数：

```
   [root@cache sbin]# /usr/local/varnish-3.0.0-beta1/sbin/varnishd -h
   /usr/local/varnish-3.0.0-beta1/sbin/varnishd: option requires an argument
-- h
   usage: varnishd [options]
   -a address:port  # HTTP listen address and port
   -b address:port  # backend address and port
    #-b <hostname or IP>
    #-b '<hostname or IP>:<port or service>'
   -C   # print VCL code compiled to C language
   -d   # debug
   -f file  # VCL script
   -F   # Run in foreground
   -h kind[,hashoptions]# Hash specification
    #   -h critbit [default]
    #   -h simple list
    #   -h classic
    #   -h classic,<buckets>
   -i identity  # Identity of varnish instance
   -l shl,free,fill # Size of shared memory file
    #   shl: space for SHL records [80m]
    #   free: space for other allocations [1m]
    #   fill: prefill new file [+]
   -M address:port  # CLI-master to connect to.
   -n dir   # varnishd working directory
   -P file  # PID file
   -p param=value   # set parameter
   -s kind[,storageoptions] # Backend storage specification
    #   -s malloc
    #   -s file  [default: use /tmp]
    #   -s file,<dir or file>
    #   -s file,<dir_or_file>,<size>
```

```
 #   -s persist{experimenta}
 #   -s file,<dir_or_file>,<size>,<granularity>
-t   # Default TTL
-S secret-file   # Secret file for CLI authentication
-T address:port  # Telnet listen address and port
-V   # version
-w int[,int[,int]]   # Number of worker threads
 #   -w <fixed_count>
 #   -w min,max
 #   -w min,max,timeout [default: -w2,500,300]
-u user  # Priviledge separation user id
```

可以看得出，不同的版本其参数多少有些不同，因此将会在下面的命令选项中分别剖析。

2. 命令选项

- -a address[:port][，address[:port][...]：该选项用于指定 Varnish 监听接收客户端请求的 IP 地址和端口号。地址的格式可以是主机名（例如，“localhost”、cache2.xx.com），也可以是以点分格式的 IPv4 形式（例如“127.0.0.1”、192.168.9.20），或者是以方括号括起的 IPv6 地址格式（例如“[::1]”）。如果没有指定具体的地址，那么 varnishd 将会监听在所有有效的 IPv4 和 IPv6 的接口上（就是网卡上所配置的 IP 上）。如果没有指定具体的端口，那么默认的 http 端口将会以“/etc/services”文件中列出的 http 对应的端口号监听（如果没有修改过该文件，那么默认的值为 80）。如果指定多个地址和端口号，那么使用空格或者是逗号将其分开列举。
- -b host[:port]：该选项用于设定使用指定的主机作为后台服务器，如果没有指定端口，那么默认值将会是 8080。
- -C：将 VCL 代码编译为 C 语言并退出。如果编译指定的是 VCL 文件，那么可以使用-f 选项指定。
- -d：该选项用于启动调试模式，父进程将会运行在前台，类似于下面的内容：

```
[root@cache sbin]#./varnishd -d  -f /usr/local/varnish-3.0.0-beta1 \
>/etc/varnish/default.vcl -s malloc,10m -T 127.0.0.1:2000 -a 0.0.0.0:8080
SMA.s0: max size 10 MB.
Platform: Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit
200 232
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
Type 'start' to launch worker process.
```

看得出它是一个基于 CLI 的命令行标准输入/输出（stdin/stdout）连接模式，如果在此时键入 help 命令，那么它的输出将会是以下内容：

```
help   //键入 help 命令
200 358
help [command]
ping [timestamp]
auth response
quit
banner
status
start
stop
stats
vcl.load <configname> <filename>
vcl.inline <configname> <quoted VCLstring>
vcl.use <configname>
vcl.discard <configname>
vcl.list
vcl.show <configname>
param.show [-l] [<param>]
param.set <param> <value>
panic.show
panic.clear
storage.list
No help from child, (not running)
```

显示的是一个命令列表，告诉我们可以使用的命令。注意最后一句，说明子进程没有启动。

此时我们在另一个窗口执行以下命令：

```
[root@cache ~]# ps -ef|grep varnish
root2605325452008:39pts/000:00:00./varnishd-d-f/usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -s malloc,10m -T 127.0.0.1:2000 -a 0.0.0.0:8080
root 26220 26018  0 08:48 pts/100:00:00 grep varnish
[root@cache ~]# lsof -i:8080
```

看得出 varnishd 已被成功执行启动，但是 8080 端口并没有打开，也就是说它并不接受连接。

下面我们再执行 start 命令：

```
start
child (26298) Started
200 0

Child (26298) said Child starts
```

执行 start 命令启动了子进程，因此在调试模式下，要想启动子进程必须使用 start 命令明确启动。

再在另外一个窗口执行以下命令：

```
[root@cache ~]# lsof -i:8080
COMMANDPID  USER   FD   TYPE DEVICE SIZE NODE NAME
varnishd 26298 nobody7u  IPv4 122168   TCP *:webcache  (LISTEN)
```

可将 8080 端口被打开，这标志着现在可以接受客户端的 HTTP 连接。同时可以 ps -ef 命令查看子进程情况。

注意 quit 命令，如果我们在这个命令界面执行该命令，那么主进程和子进程都将会退出：

```
quit
500 22
Closing CLI connection
```

- -F：将 Varnish 运行在前台。
- -f config：该选项用于指定 VCL 配置文件，而不是默认的配置文件。
- -g group：该选项用于指定一个非特权用户组，它用于子进程的运行，在 varnishd 接受连接之前，切换到子进程。
- -h type[，options]：该选项用于指定哈希算法，参考哈希算法支持的算法列表。
- -i identity：指定 Varnish 服务器的身份标识。也可以使用 VCL 中的 server.identity 指定。
- -l shmlogsize：该选项用于设置 shmlog 文件的大小，设定的数字后跟随的单位可以是 k、m，可以用到 e。默认值为 80M，如果指定的值小于 8M，那将是不明智的。
- -n name：为 Varnish 实例指定一个名字。该名字将会被用在其他情况中。例如，该名字被用于创建一个目录名字，用来存放临时文件和持久状态文件。如果指定的名字是由斜线开始，例如/cache2，那么它将会被解释为一个绝对路径，这就是“/”的作用；还可以用来获取日志，等等。如果没有指定，默认使用主机名。看下面的例子，这是默认值：

```
[root@cache varnish-3.0.0-beta1]# pwd
/usr/local/varnish-3.0.0-beta1
[root@cache varnish-3.0.0-beta1]# tree var
var
'-- varnish
'-- cache    //这是默认主机的名字
|-- _.vsm
|-- vcl.1P9zoqAU.so
'-- vcl.TcGrFssr.so

2 directories, 3 files
```

默认情况下它的名字为服务器名字 cache，位置在 Varnish 安装主目录 var/varnish 下。

- -P file：将进程的 PID 写到指定的文件中。
- -p param=value：该选项用于为指定的参数 param 设定指定的值。参考运行时参数列表，所有运行时的参数都可以通过该选项设置。该选项可以多次使用，以便指定多个参数。
- -S file：该选项用于指定一个包含密码的文件，该文件用于访问管理端口认证。

例如，我们编写一个密码文件：

```
[root@cache ~]# cat /root/pass.file
123456
```

在启动时使用该属性指定出文件的位置：

```
[root@cache ~]# /usr/local/varnish-3.0.0-beta1/sbin/varnishd -f /usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -S /root/pass.file  -s malloc,10m -T 127.0.0.1:2000 -a 0.0.0.0:80
```

> 需要注意的一点是，如果使用了该属性，那么在使用 varnishadm 命令时同样需要该属性指定同样的密码文件。

- -s type[，options]：该选项用于设定后台的存储。有关存储类型，可以参考 Varnish 所支持的存储类型。该选项可以被多次使用，以便指定多个存储文件。
- -T address[:port]：在指定的地址和端口号上提供管理接口。参考“管理接口”以便了解管理命令列表。
- -t ttl：该选项用于硬性指定缓存文档的最小生存期。也可使用 default_ttl 在运行时参数中设定。
- -u user：该选项类似于前面讲述的-g group，用于指定一个非特权用户，它用于子进程的运行，在 varnishd 接受连接之前，切换到子进程。也可使用 user 在运行时参数中设定。如果同时指定了用户和组，那么用户首先会被指定使用。
- -V：显示版本并退出。
- -w min[，max[，timeout]]：该选项用于指定 varnishd 的工作进程在指定的这个空闲时间内保持最少但又不能超过最大值的工作进程。该参数是运行时 thread_pool_min、thread_pool_max 和 thread_pool_timeout 参数的捷径设置。如果仅指定了一个数字，thread_pool_min 和 thread_pool_max 都设置为该数字，那么 thread_pool_timeout 没有效果。

3. Varnish 的存储类型

Varnish 将缓存文件存放在存储容器中，根据需要，Varnish 支持以下三种存储类型。

malloc[，size]

通过 malloc 将每一个对象都存储在内存。size 参数用于指定 varnishd 守护进程可以使用的最大内存数量，默认的单位为字节（byte），除非使用下列后缀：

- K，k：大小为 K 字节。
- M，m：大小为 M 字节。
- G，g：大小为 G 字节。
- T，t：大小为 T 字节。

参数 size 是没有大小限制的。

file[，path[，size[，granularity]]]

将每一个对象存储在后台文件中，这也是 varnishd 的默认值。

参数 path 用于指定一个文件或者是指定文件的路径，如果指定了路径，那么 varnishd 将会在该目录中创建文件，默认的目录是/tmp。

参数 size 用于指定该文件的大小，默认单位为字节（byte），除非使用下面的后缀。

- K，k：大小为 k 字节。
- M，m：大小为 M 字节。
- G，g：大小为 G 字节。
- T，t：大小为 T 字节。
- %：表示的是一个百分比值，它表示的具体含义是使用了磁盘剩余空间的百分比值。默认为 50%。如果指定的文件存在，那么它将会被截取或者扩展到指定的大小。

> 注意，如果 varnishd 不得不建立或者扩展这个文件，那么不会使用预先分配添加的空间，因为它会导致断裂（fragmentation），因此，这样可能对性能造成负面影响。预建立存储文件使用 dd 命令来实现，从而将断裂减少到最低。

参数 granularity 指定分配的粒度，所有的分配都围绕这个大小，默认单位为字节（byte），除非使用上面的后缀之一，当然除了%。默认的 size 是 VM 页的大小，如果存储的对象比较小，那么要修改该值，即减小该值。

persistence[XXX]

这是一种新的、更好的存储类型。

4. 管理接口的使用

在运行 varnishd 时，如果指定了-T 选项，那么 varnishd 将会提供一个基于命令行的接口，同时指定它监听的端口和地址。在连接到管理端口上以后就可以使用以下命令进行管理了。连接的方法可以使用 telnet，但在连接到命令行管理界面时推荐使用 varnishadm。

例如：

```
[root@cache ~]# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.
200 193
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

又如：

```
[root@cache bin]# ./varnishadm
CLI connected to 127.0.0.1 2000
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
```

```
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

在这个命令行下键入 help 命令：

```
help  //键入的命令
200 407
help [command]
ping [timestamp]
auth response
quit
banner
status
start
stop
stats
vcl.load <configname> <filename>
vcl.inline <configname> <quoted VCLstring>
vcl.use <configname>
vcl.discard <configname>
vcl.list
vcl.show <configname>
param.show [-l] [<param>]
param.set <param> <value>
panic.show
panic.clear
storage.list
ban.url <regexp>
ban <field> <operator> <arg> [&& <field> <oper> <arg>]...
ban.list
```

以上是在 3.0.0 下的命令。下面我们看一下 2.1.5 下的命令：

```
help [command]
ping [timestamp]
auth response
quit
banner
status
start
stop
stats
vcl.load <configname> <filename>
vcl.inline <configname> <quoted_VCLstring>
```

```
vcl.use <configname>
vcl.discard <configname>
vcl.list
vcl.show <configname>
param.show [-l] [<param>]
param.set <param> <value>
purge.url <regexp>
purge <field> <operator> <arg> [&& <field> <oper> <arg>]...
purge.list
```

很明显有不同，因此在介绍这些命令时尽可能地都涉及到。

管理接口可以执行的命令

下面我们分析一下这些命令。

- help [command]：该命令将返回一个有效命令的列表。
- auth response：认证响应，如果没有设置认证，那么结果是这样的：

```
auth response
300 27
Secret file not configured
```

- param.set<param><value>：为指定的参数（param）设置值（value），参考运行时参数列表。例如：

```
param.set user root  //键入的命令
200
Change will take effect when child is restarted
```

- param.show [-l] [<param>]：该命令用于显示一个运行时参数及其值。如果指定了-l 选项，那么返回的列表中包含了每个参数的简短说明。如果指定了一个具体的 param，那么仅显示该值及其对该参数的简短说明。例如：

```
param.show acceptor_sleep_decay//键入的命令
200 771
acceptor_sleep_decay   0.900000 []
   Default is 0.900
   If we run out of resources, such as file
   descriptors or worker threads, the acceptor will
   sleep between accepts.
   This parameter (multiplicatively) reduce the sleep
   duration for each succesfull accept. (ie: 0.9 =
   reduce by 10%)

   NB: We do not know yet if it is a good idea to
   change this parameter, or if the default value is
   even sensible. Caution is advised, and feedback
   is most welcome.
```

- ping [timestamp]：该命令用于 ping Varnish 缓存进程，保持连接处于活动状态。例如：

```
ping//键入的命令
200 19
PONG 1306132469 1.0
```

- purge<field><operator><arg>[&&<field><oper><arg>：该命令能够使得所有匹配清除表达式的文档无效。关于清除表达式中各部分的作用，参考下面的“清除表达式”部分。例如：

```
purge req.http.host ~ "^ (www.) example.com$" && req.url == "/cccc.jif"
200
```

- purge.list：显示清除列表。例如：

```
purge.list
200 147
0xb7e08540 1308797422.085840   420  req.url ~ ==/404.shtml
0xb7e08380 1308797391.499940   258  req.url ~ ^/xm
0xb7e08300 1308708649.657303   529G
```

- purge.url regexp：使匹配指定 URL 的所有缓存文档立即无效。例如：

```

purge.url  ^/xm
200 0

purge.url ==/404.shtml
200 0
```

- quit：退出 Varnish 管理接口。
- start：如果没有运行 Varnish 缓存的 worker 进程，那么该命令将会启动 worker 进程。
- stats：显示统计摘要。这将会展示从 Varnish 服务器从启动到现在的所有数值，如果想了解当前的形势，那么使用 varnishstat 将会更好。

例如：

```
stats//键入的命令
200 1706
  405531  Client connections accepted
  405494  Client requests received
  316040  Cache hits
   7  Cache hits for pass
   89401  Cache misses
   89458  Backend connections success
   0  Backend connections failures
   40541  Backend connections reuses
   89401  Backend connections recycles
  16  Backend connections unused
   1  N struct srcaddr
   0  N active struct srcaddr
  19  N struct sess_mem
```

```
   17  N struct sess
  281  N struct object
  280  N struct objecthead
  328  N struct smf
   34  N small free smf
   11  N large free smf
   17  N struct vbe conn
   18  N worker threads
   18  N worker threads created
    0  N worker threads not created
    0  N worker threads limited
    0  N queued work requests
   18  N overflowed work requests
    0  N dropped work requests
    89125  N expired objects
  207  N objects on deathrow
    0  HTTP header overflows
    0  Objects sent with sendfile
   342010  Objects sent with write
   405531  Total Sessions
   405501  Total Requests
   50  Total pipe
    0  Total pass
    89408  Total fetch
99672832  Total header bytes
  7266091582  Total body bytes
  405525  Session Closed
    0  Session Pipeline
    0  Session Read Ahead
    0  Session herd
19187817  SHM records
 1719720  SHM writes
1676  SHM MTX contention
    91470  allocator requests
  283  outstanding allocations
 7426048  bytes allocated
  1066315776  bytes free
    89407  Backend requests made
```

- status：该命令用于检测 Varnish 的缓存进程状态。例如：

```
status//键入的命令
200 22
Child in state running
```

- stop：该命令用于停止 Varnish 缓存的 worker 进程。例如：

```
stop
200
```

我们结合 start 命令来看一个例子，现在看 Varnish 监听的端口 8080 还在监听吗：

```
[root@cache varnish-3.0.0-beta1]# lsof -i:8080
```

可见监听端口已经被关闭，即 worker 已经停止工作。那么主进程是否在运行呢：

```
[root@cache varnish-3.0.0-beta1]# ps -ef|grep varnishd
root  7630 1  0 Jun22 ?00:00:00 /usr/local/varnish-3.0.0-beta1/
sbin/varnishd -f /usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -S
/root/pass.file -s malloc,10m -T 127.0.0.1:2000 -a 0.0.0.0:8080
www  20811  7630  0 10:40 ?00:00:00 /usr/local/varnish-3.0.0-beta1
/sbin/varnishd -f /usr/local/varnish-3.0.0-beta1/etc/varnish/default.vcl -S
/root/pass.file -s malloc,10m -T 127.0.0.1:2000 -a 0.0.0.0:8080
root 20915 26877  0 10:47 pts/000:00:00 grep varnishd
```

可见主进程仍旧在工作，因此 stop 命令只是停止了 worker 进程。我们再执行 start 命令：

```
start
200
```

看一下结果，即监听端口：

```
[root@cache varnish-3.0.0-beta1]# lsof -i:8080
COMMANDPID USER   FD   TYPE DEVICE SIZE NODE NAME
varnishd 20811  www8u  IPv4 269350   TCP *:http (LISTEN)
```

由 start 命令启动了 worker 进程。

- url.purge regexp：该命令已不建议使用，已被 purge.url 取代。
- vcl.discard<configname>：丢弃指定的配置文件 configname。如果指定的配置文件并没有被使用，那么不会有任何结果。

例如：

```
vcl.discard  default
200
```

- vcl.inline <configname> <quoted_VCLstring>：使用指定的 VCL 代码（代码字符串被引号括起）创建一个新的配置，命名为 configname。
- vcl.list：列出有效的配置，以及它们各自的被应用统计数值。活跃的配置将会用星号（*）指示出。

例如：

```
vcl.list   //键入的命令
200 15
*  19 boot
```

又如：

```
vcl.list   //键入的命令
200 23
active  2 boot
```

- vcl.load<configname><filename>：该命令用于建立一个新的配置文件，命名为

configname，但是它的内容来自于指定的文件 filename。

```
 vcl.load default  /usr/local/varnish-3.0.0-beta1/etc/varnish/
default.vcl
   200
   VCL compiled.
```

- vcl.show<configname>：显示指定配置文件的源代码。

```
vcl.show default
200
# This is a basic VCL configuration file for varnish.  See the vcl(7)
# man page for details on VCL syntax and semantics.
#
# Default backend definition.  Set this to point to your content
# server.
#
 backend ccm {
   .host = "www.xx.cn";
   .port = "80";

 }
#  //以下内容省略
```

这里需要说明的一点是，configname 指定由命令 vcl.load 载入的 configname 名字，而非 filename 名字。

- vcl.use<configname>：该命令用于使用指定的 configname 作为配置文件，对于所有新的请求都开始使用新的配置文件。现有的请求将会继续使用在它们（请求）到达时的配置文件。

```
vcl.use default
200
```

对于 configname 的说明同上。

- panic.show：该命令用于显示子进程发生恐慌或者是已经被清除的恐慌。例如：

```
panic.show   //键入的命令
300 48
Child has not panicked or panic has been cleared
```

- panic.clear：该命令用于清除已经发生的恐慌。如果没有发生过恐慌，那么清除结果将会是这样：

```
panic.clear   //键入的命令
300 17
No panic to clear
```

- storage.list：该命令用于显示存储（设备）列表。例如：

```
storage.list
200 38
Storage devices:
storage.s0 = malloc
```

- ban.url <regexp>：将所有匹配正则表达式的对象标记为过期。例如：

```
ban.url ==/m.html
200
```

- ban <field> <operator> <arg> [&& <field> <oper> <arg>]...：匹配所有指定条件的对象都标记为过期。例如：

```
ban req.http.host ~ "^(www.)example.com$" && req.url == "/cccc.jif"
200
```

- ban.list：列出仍然在活跃的被标记为过期的对象。

```
ban.list
200
0xae88a1c0  1308823611.896232  0  req.url ~ ==/mm.html
0xae88a180 1308822843.143965 1  req.http.host ~ ^(www.)example.com$ &&
req.url == /cccc.jif
```

5. 运行时参数设置

在下文的介绍中，对运行时参数做一个简短的标记，以便表明其使用情况。标记如下。

- Experimental：这种参数属于实验性参数，因此没有纯粹的好的/坏的/优化的值可指示，因此欢迎使用并反馈消息。
- delayed：参数被修改后可以在运行中改变，但是不会立即生效。
- restart：只有停止工作进程再重新启动才能使得参数生效。
- reload：修改完 VCL 参数后必须重新载入才可使得该参数生效。

这里尽可能地列出了所有参数，然而不同的版本有不同的参数，因此就不可能全部列出了，但是有一个好的方法可以去掌握你所使用版本的全部参数，那就是使用 param.show 命令，它能够罗列出现行版本中所有的参数，并且还有简短的描述。

> 另外，要注意 32 位系统，在 32 位系统上可能需要减小某些默认值的设置，例如，sess_workspace （=16k） 和 thread_pool_stack （=64k），为了保护 VM 空间。

- acceptor_sleep_decay：在资源耗尽的情况下，例如，文件描述符或者工作线程，那么接受者（acceptor）将会在接受连接之间睡眠（sleep）。对于每一个成功接受的请求，该参数以乘法的方式减少睡眠（例如 0.9 等于减少 10%）。

  默认值：0.900

  标记：experimental
- acceptor_sleep_incr：在资源耗尽的情况下，例如，文件描述符或者工作线程，那么接受者（acceptor）将会在接受连接之间睡眠（sleep）。该参数控制在每一次接受一个新连接失败后睡眠的时间的长度。

  单位：秒

默认值：0.001

标记：experimental

- acceptor_sleep_max：在资源耗尽的情况下，例如，文件描述符或者工作线程，那么接受者（acceptor）将会在接受连接之间睡眠（sleep）。该参数限制睡眠多久后才去尝试接受一个新连接。

  单位：秒

  默认值：0.050

  标记：experimental

- auto_restart：如果子进程死掉后就自动重启它。

  取值类型：bool

  默认值：on

- ban_lurker_sleep：该参数用于设置 ban lurker 线程多长时间以后（即睡眠间隔）可成功地尝试将最后一个条目推出清除列表。当没什么事可做的时候，它总是睡眠一秒钟。如果设置为 0，那么将会禁用 ban lurker 线程。

  单位：秒

  默认值：0.0

- between_bytes_timeout：该参数用于设置从后台获取数据时两个自己之间默认超时间隔。在放弃传输之前会等待该参数设定的时间，即如果超时，那么数据将被放弃传输。当将该值设置为 0 时，那么表示不会发生超时。对于每一个后端请求和后台请求，VCL 的设置能够将这些默认设置覆盖掉。需要注意的一点是该参数不能应用到 pipe。

  单位：秒

  默认值：60

- cache_vbe_conns：缓存 vbe_conn 或者是依赖于 malloc，是一个问题。

> 注意：现在还仍然不能确定将该参数设为 on 是不是一个好的想法，或者使用默认值是一个明智的选择。对于测试的结果哪一个更好欢迎反馈给开发者。

  取值类型：bool

  默认值：off

  标记：experimental

- cc_command：该参数设置的命令用于将 C 源代码编译为 dlopen 可载入的对象。任何在字符串中出现的%s 将会被源文件名称替代，而%o 也将会被输出文件名称替代。注意，该参数不能立即生效，直到 VCL 被重新载入。

  默认值：exec cc -fpic -shared -Wl，-x -o %o %s

  标记：must_reload

- cli_buffer：该参数用于设定命令行（CLI）输入缓存大小。如果是大的 VCL 文件或者是使用 vcl.inline 命令行命令，那么就要适当地增加该值。注意，必须指定-p 来生效。

  单位：字节

默认值：8192

- cli_timeout：该参数用于设置子进程从命令行请求主进程回复的超时设置。
  单位：秒
  默认值：10
- clock_skew：该参数用于设定一个时钟差，及在后台和 Varnish 服务器所在的机器时钟之间的时间差。
  单位：秒
  默认值：10
- connect_timeout：该参数用于设定与后台连接的超时设置。在这个设定的时间内 Varnish 的工作进程会不断地尝试连接后台服务器，直到超时放弃尝试连接。对于每一个后台，及对后台的请求，VCL 配置文件中的设置可以覆盖掉该默认值。
  单位：秒
  默认值：0.4
- default_grace：该参数用于提供一个默认的宽限期（grace period）。当一个对象在生存期已过后，通过该参数为它提供一个宽限期（就是一定的时间长度），以便提供其他线程尝试获取一个新的复制。
  单位：秒
  默认值：10
- default_ttl：如果后台和 VCL 代码都没有给对象指定缓存时间，那么该参数的设置值将会被使用。对于已经缓存的对象，该值的改变将不会影响到已经缓存的对象，直到它们被从后台服务器再次获取才会使用修改后的配置值。在某些情况下，如果想强迫设置立即使其生效，那么可以使用“purge.url .”刷新所有缓存对象。
  单位：秒
  默认值：120
- diag_bitmap：控制代码表示。
  单位：bitmap
  默认值：0

bitmap 控制诊断代码：

```
0x00000001 - CNT_Session states.
0x00000002 - workspace debugging.
0x00000004 - kqueue debugging.
0x00000008 - mutex logging.
0x00000010 - mutex contests.
0x00000020 - waiting list.
0x00000040 - object workspace.
0x00001000 - do not core-dump child process.
0x00002000 - only short panic message.
0x00004000 - panic to stderr.
0x00008000 - panic to abort2 () .
```

```
0x00010000 - synchronize shmlog.
0x00020000 - synchronous start of persistence.
0x80000000 - do edge-detection on digest.
```

- err_ttl：设定分配给合成错误页面的 TTL 值。
  单位：秒
  默认值：0
- esi_syntax：控制代码表示。
  单位：bitmap
  默认值：0

bitmap 控制 ESI 解析代码:

```
0x00000001 - Don't check if it looks like XML
0x00000002 - Ignore non-esi elements
0x00000004 - Emit parsing debug records
```

- fetch_chunksize：设定默认的 chunksize 大小。
  单位：KB
  默认值：128
  标记：experimental
- first_byte_timeout：该参数用于设置从后台接收第一个字节的超时情况。在连接后台服务器时，放弃一个连接之前需要等待的时间。如果将该值设置为 0，那么意味着从不超时。对于每一个后端请求和后台请求，VCL 的设置能够将这些默认设置覆盖掉。需要注意的一点是，该参数不能应用到 pipe。
  单位：秒
  默认值：60
- group：该参数用于设置运行 Varnish 服务器的非特权组。
  标记：must_restart
- http_headers：可以处理 HTTP 头的最大数目。
  单位：header line
  默认值：64
- http_range：启用对 HTTP range 头的支持，这是一个实验性的支持，使得 Varnish 为客户端提供对象的一部分。然而，Varnish 仍将从后台请求整个对象。
  默认值：off
- listen_address：该参数用于设定 Varnish 接收监听请求的地址，它的值是一个以空格分隔的列表，可能使用的格式：host，host:port，:port。
  默认值：80
  标记：must_restart
- listen_depth：该参数用于设定监听队列的深度。注意，该参数只有在子进程重启后才会生效。

单位：connection（连接）

默认值：1024

标记：must_restart

- log_hashstring：将哈希字符串记录到共享内存日志。

  取值类型：bool

  默认值：off

- log_local_address：在 SessionOpen 共享内存记录中记载 TCP 连接的本地地址。

  取值类型：bool

  默认值：off

- lru_interval：该参数用于设置将一个对象移动到 LRU 列表（最近最少使用列表）之前的时间间隔（或者叫宽限期）。如果在设定的超时时间段内这些对象没有被移动，那么这些对象仅被移动到 LRU 列表的前面。对于 LRU 列表的访问来说，这将会减少锁操作数量。

  单位：秒

  默认值：2

  标记：experimental

- max_esi_includes：该参数用于设定 esi:include 处理的最大深度。

  单位：包含数

  默认值：5

- max_restarts：该参数用于设定一个请求可以重新启动（restart）的最大次数，即上限。

> 注意，重新启动可能会引起后端服务器问题，因此不要想当然地增加该值。

  单位：重启数

  默认值：4

- overflow_max：该参数用于设置允许溢出队列长度的百分比。它设置了请求排队与工作线程的比例，超出去的会话将会被丢弃而不是排队。

  单位：%（百分比）

  默认值：100

  标记：experimental

- ping_interval：该参数用于设定从父进程到子进程 ping 的时间间隔。如果设置为 0，那么将会完全禁止 ping。

> 注意。修改该参数后可能需要重新启动子进程后才有效。

  单位：秒

  默认值：3

  标记：must_restart

- pipe_timeout：该参数用于设定 PIPE 会话超时，如果在这段时间内双方都没有收到连接信号（或者称之为流量或请求），那么会话被关闭。

单位：秒

默认值：60

- prefer_ipv6：当连接既提供了 IPv4，又提供了 IPv6 的地址时，可以通过该参数设置使用 IPv6 来连接后台服务器。

  单位：bool

  默认值：off

- purge_dups：检查并消除重复的清除（purge）。

  取值类型：bool

  默认值：on

- saintmode_threshold：该参数用于设定在 saint 模式下挡住访问对象的最大值。如果设置为 0，那么将会禁用 saint 模式。

  单位：对象（object）

  默认值：10

  标记：experimental

- send_timeout：该参数用于设定客户端连接发送超时。如果在这个设定的时间内没有数据发送到客户端，那么会话就会被关闭。

  单位：秒

  默认值：600

  标记：delayed

- sendfile_threshold：该参数用于设定通过 sendfile 发送对象的最小值。

  单位：字节

  默认值：-1

  标记：experimental

- sess_timeout：该参数用于设定持久会话的空闲超时。如果一个 HTTP 请求在这个设定的时间内仍然没有收到响应，那么该会话将会被关闭。

  单位：秒

  默认值：15

- sess_workspace：该参数用于为与会话相关的 HTTP 协议字节设定工作空间，这个空间必须足够大以便能够容纳下整个 HTTP 协议头，而且它们中的任何一个头都可能会在 VCL 代码中编辑。

  单位：字节

  默认值：65536

  标记：delayed

- session_linger：在先前请求被处理完之后，差不多一半的会话在前 100 毫秒之内会被重新利用，如果会话被重新利用，即如果一个新的请求出现，那么响应这个请求的工作线程存在多久。对于本参数，如果设置得过高，那么会导致工作线程什么都不做而在那里保存着，造成资源浪费；如果设置得太低，那么意味着更多的会话在 waiter 上走弯路，

换句话说，就是劳驾 waiter 重新分配资源，这同样会浪费资源。

单位：毫秒

默认值：50

标记：experimental

- session_max：该参数用于设定会话的最大值，如果超过这个值，那么将会丢弃连接。这个设置通常是防止 DoS 攻击，将其设置得足够高将不会受到伤害，只要内存足够。

  单位：sessions

  默认值：100000

- shm_reclen：该参数用于设定 SHM 中记录的最大长度，最大值为 65535 字节。

  单位：字节

  默认值：255

- shm_workspace：该参数用于设定 shmlog 工作区分配给工作线程的空间大小。如果设置的太大，那么将会浪费一些 RAM；如果设置得太小，那么会导致不必要的将 RAM 中的内容冲出，发生被冲出情况时，将会被记录到"SHM flushes due to overflow"状态中。最小值为 4096 字节。

  单位：字节

  默认值：8192

  标记：delayed

- syslog_cli_traffic：设置了该参数后，所有的 CLI 流量都将会记录到 syslog，即系统日志。

  取值类型：bool

  默认值：no

- thread_pool_add_delay：该参数用于设定添加线程的延时值，换句话说，就是创建两个线程之间至少要等待这么长时间。如果将该延时设置得太长，可能会导致工作线程不足；如果将该值设置得太短，则可能增加工作进程堆积的风险。

  单位：millisecond （毫秒）

  默认值：20

  标记：experimental

- thread_pool_add_threshold：该参数用于设定创建工作进程的溢出门限值。如果将该值设置得太低，那么将导致工作线程过多，这种做法通常是不可取的；如果设置得过高，那么又会出现工作进程不足。

  单位：request（请求）

  默认值：2

  标记：experimental

- thread_pool_fail_delay：该参数用于设定一个延时时间。当创建一个线程失败以后，至少要等待这个时间之后才再次尝试创建另一个进程。因为进程运行用完存放线程堆栈的 RAM 资源，在创建工作进程失败时经常会在堆栈底部做一个标记。设置该延时值就是试图不将该值不必要的写入。如果创建线程失败成为一个问题，那么就要检查

thread_pool_max 是不是太高。增加 thread_pool_timeout 和 thread_pool_min 的值也可以帮助减少线程的消亡和重建率。

单位：毫秒

默认值：200

标记：experimental

- thread_pool_max：该参数用于设定最大的线程数，它设置的是所有线程池的总和。由于过多的工作线程会迅速地占用 RAM 和 CPU 资源，因此，在设置这个值时，不要设置得太高，通常设置为能够正常完成给它的工作正好。

  单位：thread（线程）

  默认值：500

  标记：delayed，experimental

- thread_pool_min：该参数用于设定每一个线程池最小的线程数量。增加该值有助于在已有的线程生存期过后，当出现大量的请求时能够有个从低负荷到高负荷提高的缓冲梯度。最小值为 2 个线程。

  单位：thread（线程）

  默认值：5

  标记：delayed，experimental

- thread_pool_purge_delay：该参数用于设定在清除线程之前的延时时间，及空闲时间已过后的延时。最小值为 100 毫秒。

  单位：毫秒

  默认值：1000

  标记：delayed，experimental

- thread_pool_stack：该参数用于设定线程堆栈的大小。特别是在 32 位的系统中，需要调整这个值，以便使得大量的线程适合有效的地址空间。

  单位：byte（字节）

  默认值：-1

  标记：experimental

- thread_pool_timeout：该参数用于设置一个超时值，当现有的线程超出参数 thread_pool_min 设定的数量，但这些线程已经处于空闲状态，通过该参数为这些空闲的线程设置一个空闲超时值，一旦超过这个值，则会被清除。最小值为 1 秒。

  单位：秒

  默认值：300

  标记：delayed，experimental

- thread_pools：该参数的功能在于设定工作线程池（worker thread pool）数量。增加工作线程池将会减少锁竞争。过多的 pool 会浪费 CPU 和 RAM 资源，每个 CPU 多于一个 pool，将会影响性能。增加 pool 可以在运行中添加，而减少则需要重新启动才会生效。

  单位：pool

默认值：2

标记：delayed，experimental

- thread_stats_rate：worker 线程累积统计。当它们完成一个请求后，如果锁被释放，那么这些状态将会被转储到全局状态计数器。该参数定义了一个 worker 线程可以处理的最大请求数，在它被强制转储之前，它的堆积统计被存储在全局计数器中。

  单位：request

  默认值：10

  标记：experimental

- user：该参数用于设定一个非特权的用户来运行 Varnish。如果设定用户，那么也需要设定该用户所在的组“group”。

  标记：must_restart

- vcl_trace on：在 shmlog 中追踪 VCL 执行。如果开启该功能，那么将会看到每一个通过 VCL 程序请求的路径，这样会产生大量的日志记录，因此该功能默认是被关闭的。

  取值类型：bool

  默认值：off

- waiter：选择内核接口，默认为 default，default 指的是 epoll，poll。

6. 清除表达式

清除表达式包括一个或者是多个条件，条件由字段、操作符和参数组成，多个条件可以通过“&&”进行组合。

- 字段：可以是来自于 VCL 中的任何一个变量，例如 req.url、req.http.host 或 obj.set-cookie。
- 操作符：“==”用于直接比较，“~”用于正则表达式匹配，“>”或“<”用于大小比较，“!”是否定操作符。
- 参数：可以是一个由双引号括起的字符串、一个表达式或者是一个整数，对于整数会有单位，单位可以是“KB”，“MB”，“GB”或“TB”，将其附加在大小值的后面。

看下面的三个例子：

- 清除准确匹配/logo.gif 字符串的缓存文档：

```
req.url == "/logo.gif"
```

- 清除所有的不以“.ogg”结尾并且大小超过 10MB 的所有缓存文档：

```
req.url !~ "\.ogg$" && obj.size > 10MB
```

- 清除所有主机名为“example.com”或 www.example.com 并且 Set-Cookie 头包含“USERID=1663”的所有缓存文档：

```
req.http.host ~ "^ (www.) example.com$" && obj.set-cookie ~ "USERID=1663"
```

这三个例子来自于官方文档，对不同的 Varnish 版本可能执行效果不同，就是说可能不会成功执行，要根据具体的版本修改。

7. 查看状态信息

在成功地安装配置并运行了 Varnish 服务器后，可以通过 Varnish 服务器提供的一些工具来

查看它的状态信息。这些命令在安装后的目录 bin 中，例如 varnishtop、varnishhist，等等，参考命令部分。

这里的命令就是 sbin 和 bin 下的可执行命令，其中 varnishd 命令为 Varnish 的守护进程，而其他的命令有的用于监控，有的可用于设置参数等。

57.11 Varnish 提供的命令

Varnish 提供了以下命令用于管理、查看和测试 Varnish 服务器。

1. varnishadm

命令行工具 varnishadm 通过使用-T 和-S 来连接 varnishd 进程。用于管理运行中的 Varnish 实例。如果命令成功执行。那么它退出的状态代码为零，否则状态代码为非零。

如果在 varnishadm 命令行中给定了命令和参数，那么给定的命令和参数将会发送到 varnishd 进程并且在标准的输出设备（即显示器屏幕）上会有输出。

例如：

```
[root@cache bin~]# ./varnishadm  -T 127.0.0.1:2000  status
CLI connected to 127.0.0.1:2000
Child in state running
```

如果在命令行中没有指定命令参数，那么将会进入一个交互的界面，标准的输入设备为键盘，标准的输出设备为显示器。

例如：

```
[root@cache ~]# ./varnishadm  -T 127.0.0.1:2000
CLI connected to 127.0.0.1:2000
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

如果出现以下情况：

```
[root@mail bin]# ./varnishadm -T 127.0.0.1:6082
Authentication required
Write error CLI socket: Bad file descriptor
```

注意黑体字部分，说明需要认证。再看以下连接：

```
[root@mail bin]# ./varnishadm -T 127.0.0.1:6082 -S /root/pass.file
CLI connected to 127.0.0.1:6082
```

```
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

黑体字部分为密码认证文件。

语法格式：

```
[root@cache bin]# ./varnishadm  -h
./varnishadm: invalid option -- h
usage: varnishadm [-n ident] [-t timeout] [-S secret file]
         -T [address]:port command [...]
   -n is mutually exlusive with -S and -T
```

varnishadm 参数如下。

- -t timeout：该参数用于设定操作超时，一旦超过这个时间就不再等待了。
- -S secret_file：该参数用于指定认证密钥文件。要使用这个参数的前提是在 Varnish 启动时使用-S 参数指定了密码文件。只有读取该文件内容通过认证的进程才能够进行命令行连接。

例如：

```
[root@cache~]#/usr/local/varnish-3.0.0-beta1/bin/varnishadm-T127.0.0.1:2
000
Authentication required
Write error CLI socket: Bad file descripto
```

可以看得出，我们的连接没有成功，原因在于它的管理需要认证。因此，需要使用该参数指出认证文件的位置：

```
[root@cache~]#/usr/local/varnish-3.0.0-beta1/bin/varnishadm-T127.0.0.1:2
000  -S pass.file
CLI connected to 127.0.0.1:2000
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

看一下在 varnishadm 中可以执行的命令：

```
Help    //敲入的命令
200
help [command]  //以下为可使用的命令
ping [timestamp]
auth response
quit
banner
status
start
stop
stats
vcl.load <configname> <filename>
vcl.inline <configname> <quoted VCLstring>
vcl.use <configname>
vcl.discard <configname>
vcl.list
vcl.show <configname>
param.show [-l] [<param>]
param.set <param> <value>
panic.show
panic.clear
storage.list
ban.url <regexp>
ban <field> <operator> <arg> [&& <field> <oper> <arg>]...
ban.list
```

- -T [address]:port：该参数用于指定连接 varnishd 管理接口的地址和端口号。
- Command：可用的命令和参数，可以参考 varnishd 命令部分。可以通过 help 命令来获取一个可以使用的命令列表，可以使用 param.show 命令来显示参数的简单概要。

例如：通过 param.show 命令显示所有的运行时变量。

```
[root@cache bin]# ./varnishadm  -T 127.0.0.1:2000
CLI connected to 127.0.0.1:2000
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.

param.show
200
acceptor_sleep_decay   0.900000 []
```

```
acceptor_sleep_incr0.001000 [s]
acceptor_sleep_max 0.050000 [s]
auto_restart  on [bool]
ban dups  on [bool]
ban lurker sleep  0.010000 [s]
between bytes timeout  60.000000 [s]
cc command "exec gcc -std=gnu99  -pthread -fpic -shared -Wl,-x -o %o %s"
cli_buffer 8192 [bytes]
cli_timeout10 [seconds]
clock_skew 10 [s]
connect_timeout0.700000 [s]
critbit cooloff180.000000 [s]
default grace  10.000000 [seconds]
default keep  0.000000 [seconds]
default ttl120.000000 [seconds]
diag_bitmap0x0 [bitmap]
esi_syntax 0 [bitmap]
expiry_sleep  1.000000 [seconds]
fetch chunksize128 [kilobytes]
fetch maxchunksize 262144 [kilobytes]
first byte timeout 60.000000 [s]
group  nobody (99)
gzip_level 6 []
gzip_stack_buffer  4096 [Bytes]
gzip_tmp_space 0 []
http_gzip_support  on [bool]
http max hdr  64 [header lines]
http range support on [bool]
http req hdr len  2048 [bytes]
http req size  32768 [bytes]
http_resp_hdr_len  2048 [bytes]
http_resp_size 8192 [bytes]
listen_address 0.0.0.0:8080
listen depth  1024 [connections]
log hashstring on [bool]
log local address  off [bool]
lru interval  2 [seconds]
max_esi_depth  5 [levels]
max_restarts  4 [restarts]
ping_interval  3 [seconds]
pipe_timeout  60 [seconds]
prefer ipv6off [bool]
queue_max  100 [%]
```

```
rush_exponent  3 [requests per request]
saintmode_threshold10 [objects]
send_timeout   60 [seconds]
sess timeout   5 [seconds]
sess workspace 16384 [bytes]
session linger 50 [ms]
session max100000 [sessions]
shm_reclen 255 [bytes]
shm_workspace  8192 [bytes]
shortlived 10.000000 [s]
syslog_cli_traffic on [bool]
thread pool add delay  2 [milliseconds]
thread pool add threshold  2 [requests]
thread pool fail delay 200 [milliseconds]
thread pool max500 [threads]
thread_pool_min5 [threads]
thread_pool_purge_delay1000 [milliseconds]
thread_pool_stack  65536 [bytes]
thread pool timeout300 [seconds]
thread pool workspace  16384 [bytes]
thread pools   2 [pools]
thread stats rate  10 [requests]
user   nobody (99)
vcc_err_unref  on [bool]
vcl_dir/usr/local/varnish-3.0.0-beta1/etc/varnish
vcl_trace  off [bool]
vmod dir   /usr/local/varnish-3.0.0-beta1/lib/varnish/vmods
waiter default (epoll, poll)
```

又如：通过 param.show 命令来显示某一个变量。

```
param.show group
200
group  nobody (99)
   Default is r
   The unprivileged group to run as.

   NB: This parameter will not take any effect until
   the child process has been restarted.
```

更多内容参考 varnishd 命令。

2. varnishncsa

varnishncsa 工具读取 varnishd 的共享日志，并且以 Apache / NCSA “combined” 的日志格式显示它们。

例如：

```
[root@s8 bin]# ./varnishncsa
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/images/index_images/new/new.gif HTTP/1.0" 200 154 "http://www.xx.cn/" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/lib/css/news/news lastpage.cssHTTP/1.0"3040"http://happy.xx.cn/346/c/201106/24/n3383639,4.shtml" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; QQDownload 677; SV1; 4399Box.560) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/images/ad/vip_2.gifHTTP/1.0"304 0 "http://happy.xx.cn/346/c/201106/24/n3383603.shtml" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/lib/css/news/news lastpage.cssHTTP/1.0"3040"http://news.xx.cn/351/c/201106/23/n3382793,3.shtml" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/images/index_images/new/sjdy.gif HTTP/1.0" 200 406 "http://www.xx.cn/" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/lib/css/news/news lastpage.cssHTTP/1.0"3040"http://happy.xx.cn/346/c/201106/23/n3382743,10.shtml" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1) "
127.0.0.1--[25/Jun/2011:10:49:28+0800]"GEThttp://127.0.0.1:8080/images/index_images/new/daoying.png HTTP/1.0" 200 2190 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; GTB6.5); QQBrowser/5.1.7061.201 (trident) "
```

语法格式：

```
[root@cache ~]# ./varnishncsa -h
./varnishncsa: invalid option -- h
usage: varnishncsa [-bCcd] [-i tag] [-I regexp] [-k keep]
             [-m tag:regex] [-n varnish name] [-r file] [-s skip]
             [-X regexp] [-x tag] [-aDV]
             [-n varnish_name] [-P file] [-w file]
```

下面我们看一下这些参数。

- -a：在向一个文件写入时采用添加的方法而不是将其覆盖。
- -b：包含同后台服务器通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。
- -C：当使用正则表达式匹配时，忽略大小写。
- -c:包含同客户端通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。
- -D：以守护进程的方式运行 varnishlog。
- -d：在 varnishlog 启动时处理旧的日志条目。通常情况下，varnishlog 仅处理在它启动之后生成的日志。

- -f：在输出日志中使用 X-Forwarded-For HTTP 头替代 client.ip。
- -I regex：只处理匹配指定正则表达式的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -i tag：处理指定 tag 的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -n：该参数用于指定 varnishd 实例的名字，以便 varnishhist 工具处理指定实例的日志。如果没有-n 参数，那么当前物理服务器的名字将会被使用。
- -P file：将进程的 PID 文件写入指定的文件。
- -r file：从指定的日志文件中读取日志条目，而不是从共享内存中读取。
- -V：显示版本号并退出。
- -w file：将日志条目写入文件而不是显示它们。如果指定的文件存在且没有使用-a 选项，那么原有文件将会被覆盖。当 varnishlog 正在向文件中写入的时候，如果 varnishlog 收到一个 SIGHUP，那么 varnishlog 将会重新打开该文件，并且原有的文件将会被旋转（rotate）。
- -X regex：通过匹配指定的正则表达式来排除不处理的日志条目。
- -x tag：通过指定 tag 来排除不处理的日志条目。

3. varnishhist

varnishhist 工具通过读取 varnishd 共享内存日志，并且以柱状图的形式持续不断地更新，显示了最后 N 个被处理的请求。图的左上角有 N 的值和纵坐标（垂直）缩放比例；横坐标（水平）方向的缩放是以对数形式呈现。我们在图中还看到有两种符号，一种是管道符（“|”），而另一种是哈希字符（“#”），它们代表着不同的意义：如果被命中（hit）那么就会标记为“|”，而没有命中的则标记为“#”。这些在“查看状态信息”部分有描述。

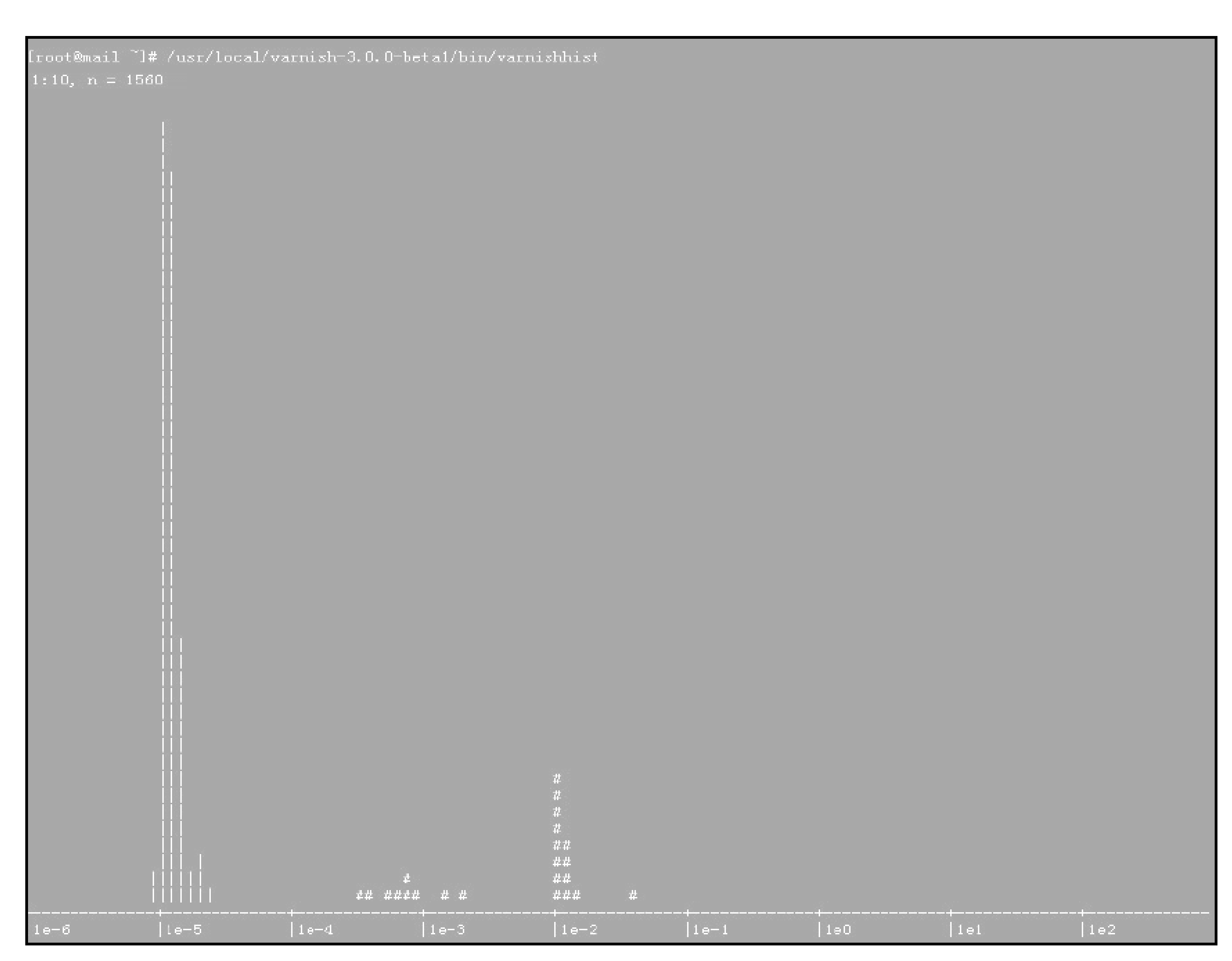
语法格式：

```
[root@cache bin]# ./varnishhist -h
./ varnishhist: invalid option -- h
usage: varnishhist [-bCcd] [-i tag] [-I regexp] [-k keep]
                  [-m tag:regex] [-n varnish_name] [-r file]
                  [-s skip] [-X regexp] [-x tag] [-n varnish name]
                  [-V] [-w delay]
```

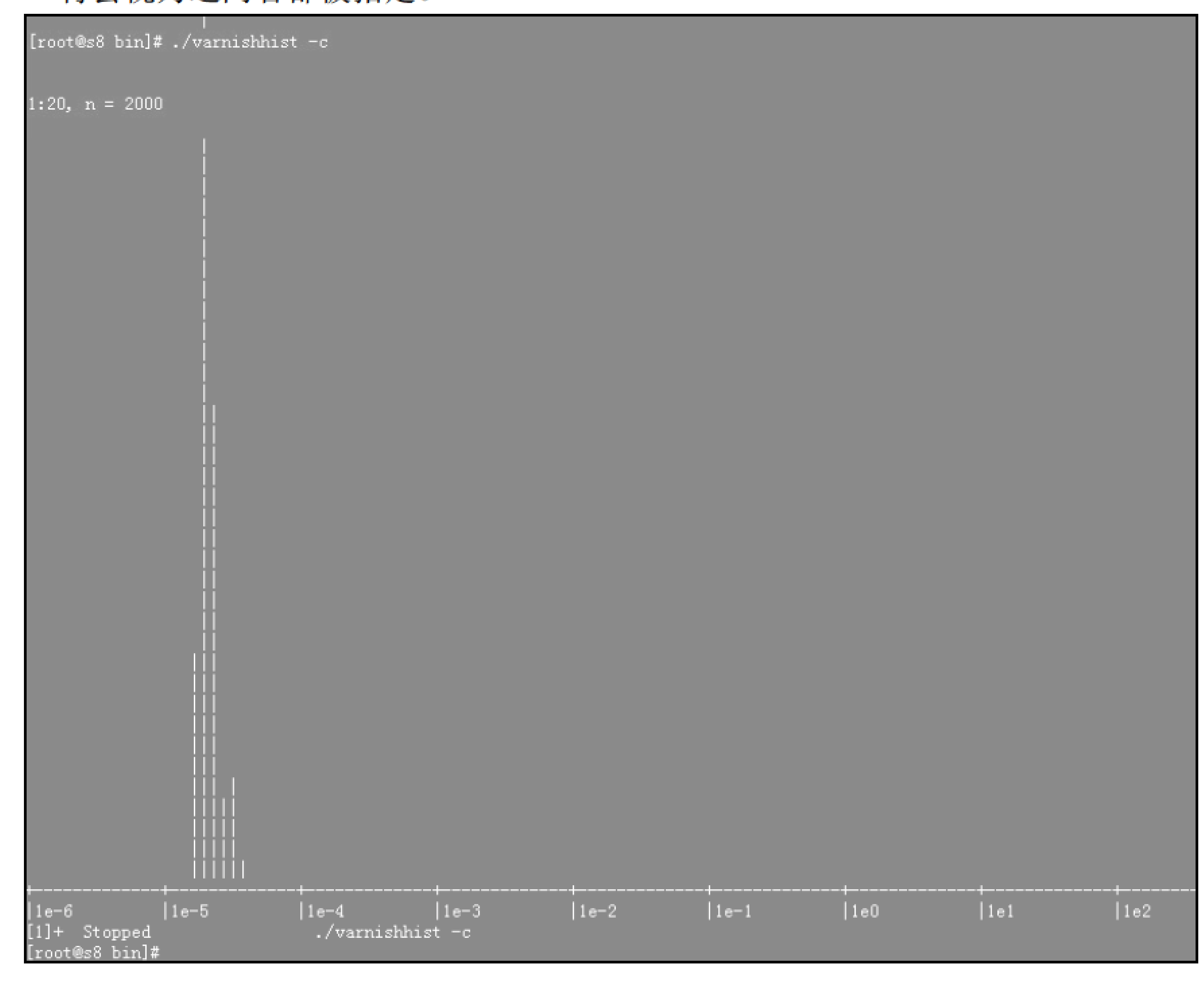
下面我们看一下这些参数：

- -b：包含同后台服务器通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。

例如：



- -C：当使用正则表达式匹配时，忽略大小写。
- -c：包含同客户端通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。



- -d：在 varnishhist 启动时处理旧的日志条目。通常情况下，varnishhist 仅处理在它启动之后生成的日志。
- -I regex：只处理匹配指定正则表达式的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -i tag：处理指定 tag 的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -n：该参数用于指定 varnishd 实例的名字，以便 varnishhist 工具处理指定实例的日志。如果没有-n 参数，那么当前物理服务器的名字将会被使用。
- -r file：从指定的日志文件中读取日志条目，而不是从共享内存中读取。
- -V：显示版本号并退出。

```
[root@cache bin]# ./varnishhist  -V
varnishhist (varnish-3.0.0-beta1 revision varnish-3.0.0-beta1)
Copyright (c) 2006-2009 Linpro AS / Verdens Gang AS
```

- -w delay：该参数用于设定每一次升级间隔，默认值为 1 秒。
- -X regex：通过匹配指定的正则表达式来排除不处理的日志条目。
- -x tag：通过指定 tag 来排除不处理的日志条目。

4. varnishlog

varnishlog 工具是读取和显示 varnishd 的共享内存日志。

语法格式：

```
[root@cache bin]# ./varnishlog -h
./varnishlog: invalid option -- h
usage: varnishlog [-bCcd] [-i tag] [-I regexp] [-k keep]
            [-m tag:regex] [-n varnish_name] [-r file]
            [-s skip] [-X regexp] [-x tag] [-aDV] [-o [tag regex]]
            [-n varnish_name] [-P file] [-w file]
```

下面我们看一下这些参数：

- -a：在向一个文件写入时采用添加的方法而不是将其覆盖。
- -b：包含同后台服务器通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。例如：

```
[root@s8 bin]# ./varnishlog
  12 SessionOpen  c 127.0.0.1 45818 0.0.0.0:8080
  11 SessionOpen  c 127.0.0.1 45817 0.0.0.0:8080
  12 ReqStart c 127.0.0.1 45818 1192728209
  12 RxRequestc GET
  12 RxURLc /images/images home/3/bm 0.jpg
  12 RxProtocol   c HTTP/1.0
  12 RxHeader c Host: 127.0.0.1:8080
  12 RxHeader c Connection: close
  12 RxHeader c Accept: */*
```

```
    12 RxHeader c Referer: http://news.xx.cn/351/c/201106/25/n3384483.shtml
    12 RxHeader c Accept-Language: zh-cn
    12 RxHeader c Accept-Encoding: gzip, deflate
    12 RxHeader c User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1; .NET CLR 2.0.50727; AskTbPTV/5.11.3.15590)
    12 VCL call c recv
    12 VCL return   c lookup
    12 VCL_call c hash
    12 VCL_return   c hash
    12 Hit  c 1192709761
    12 VCL_call c hit
    12 VCL return   c deliver
    12 VCL call c deliver
    12 VCL return   c deliver
    12 TxProtocol   c HTTP/1.1
    12 TxStatus c 200
    12 TxResponse   c OK
    12 TxHeader c Server: Nginx/0.8.53 (Unix)
    12 TxHeader c Last-Modified: Thu, 05 Jun 2008 08:27:39 GMT
    12 TxHeader c ETag: "574024-2a9b-1e4394c0"
    12 TxHeader c Content-Type: image/jpeg
    12 TxHeader c Content-Length: 10907
    12 TxHeader c Date: Sat, 25 Jun 2011 02:23:26 GMT
    12 TxHeader c X-Varnish: 1192728209 1192709761
    12 TxHeader c Age: 102
    12 TxHeader c Via: 1.1 varnish
    12 TxHeader c Connection: close
```

- -C：当使用正则表达式匹配时，忽略大小写。
- -c：包含同客户端通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishhist 将会视为这两者都被指定。
- -D：以守护进程的方式运行 varnishlog。
- -d：在 varnishlog 启动时处理旧的日志条目。通常情况下，varnishlog 仅处理在它启动之后生成的日志。
- -I regex：只处理匹配指定正则表达式的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -i tag：处理指定 tag 的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。

例如：

```
[root@s8 bin]# ./varnishlog -i TxHeader -I ^Age
    11 TxHeader c Age: 5
    11 TxHeader c Age: 9
```

```
    11 TxHeader c Age: 67
    11 TxHeader c Age: 9
    11 TxHeader c Age: 9
    11 TxHeader c Age: 9
    12 TxHeader c Age: 9
    11 TxHeader c Age: 9
    11 TxHeader c Age: 9
    11 TxHeader c Age: 6
    11 TxHeader c Age: 9
    11 TxHeader c Age: 9
    11 TxHeader c Age: 14
    11 TxHeader c Age: 13
    11 TxHeader c Age: 62
    11 TxHeader c Age: 104
    11 TxHeader c Age: 102
    11 TxHeader c Age: 12
    11 TxHeader c Age: 14
    11 TxHeader c Age: 63
    11 TxHeader c Age: 14
    12 TxHeader c Age: 13
    11 TxHeader c Age: 0
    11 TxHeader c Age: 59
    11 TxHeader c Age: 117
    11 TxHeader c Age: 104

...
```

> 说明一下，Varnish 添加了一个 Age 头，以便来指示说明一个对象在 Varnish 中存储了多久，在这个例子中我们通过 varnishlog 命令输出它们。

- -k num：仅显示前 num 条记录。
- -m tag:regex：只处理列出的与 regex 匹配的 tag。多个-m 选项将被合并在一起。该选项不能和-O 组合使用。
- -n：该参数用于指定 varnishd 实例的名字，以便 varnishhist 工具处理指定实例的日志。如果没有-n 参数，那么当前物理服务器的名字将会被使用。
- -o：忽略掉早期版本的兼容性。
- -O：不通过请求 ID 将日志条目分组。该选项不能和-m 组合使用。

```
[root@s8 bin]# ./varnishlog -c -o ReqStart 192.168.4.7
    11 SessionOpen  c 192.168.4.7 4104 0.0.0.0:8086
    11 ReqStart c 192.168.4.7 4104 646176274
    11 RxRequestc GET
    11 RxURLc /images/index_images/new/ico_sina.gif
    11 RxProtocol   c HTTP/1.1
```

```
      11 RxHeader c Host: 192.168.4.8:8086
      11 RxHeader c User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.2.14) Gecko/20110218 Firefox/3.6.14
      11 RxHeader c Accept: image/png,image/*;q=0.8,*/*;q=0.5
      11 RxHeader c Accept-Language: en-us,en;q=0.5
      11 RxHeader c Accept-Encoding: gzip,deflate
      11 RxHeader c Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
      11 RxHeader c Keep-Alive: 115
      11 RxHeader c Connection: keep-alive
      11 RxHeader c Referer: http://192.168.4.8:8086/

    …
```

- -P file：将进程的 PID 文件写入指定的文件。
- -r file：从指定的日志文件中读取日志条目，而不是从共享内存中读取。
- -s num：跳过前 num 个日志记录。
- -u：不缓存输出。
- -V：显示版本号并退出。
- -w file：将日志条目写入文件而不是显示它们。如果指定的文件存在且没有使用-a 选项，那么原有文件将会被覆盖。当 varnishlog 正在向文件中写入的时候，如果 varnishlog 收到一个 SIGHUP，那么 varnishlog 将会重新打开该文件，并且原有的文件将会被滚动命名，即将现有的日志文件按照规律重命名而再建立一个同原来文件名相同的文件，原有 filename 变为 filename1，原有的 filename1 变为 filename2……然后再重新生成 filename。
- -X regex：通过匹配指定的正则表达式来排除不处理的日志条目。
- -x tag：通过指定 tag 来排除不处理的日志条目。

如果指定了-o 选项，那么需要指定一个额外的 tag 和 regex，以便仅选择匹配指定 regex 的日志。

下列是有效的 tag：

· Backend
· BackendClose
· BackendOpen
· BackendReuse
· BackendXID
· CLI
· ClientAddr
· Debug
· Error
· ExpBan
· ExpKill
· ExpPick

- Hit
- HitPass
- HttpError
- HttpGarbage
- Length
- ObjHeader
- ObjLostHeader
- ObjProtocol
- ObjRequest
- ObjResponse
- ObjStatus
- ObjURL
- ReqEnd
- ReqStart
- RxHeader
- RxLostHeader
- RxProtocol
- RxRequest
- RxResponse
- RxStatus
- RxURL
- SessionClose
- SessionOpen
- StatAddr
- StatSess
- TTL
- TxHeader
- TxLostHeader
- TxProtocol
- TxRequest
- TxResponse
- TxStatus
- TxURL
- VCL_acl
- VCL_call
- VCL_return
- VCL_trace
- WorkThread

5. varnishreplay

varnishreplay 工具用于解析 Varnish 的日志，并且试图重现流量。它的功能通常用来热身缓

存或各种形式的测试。

语法格式：

```
[root@cache bin]# ./varnishreplay -h
./varnishreplay: invalid option -- h
usage: varnishreplay [-D] -a address:port -r logfile
```

- -a backend：将这些流量通过 TCP 发送到后台，通过地址和端口号来实现。该选项为强制选项。目前仅支持 IPv4。
- -D：打开调试模式。
- -r file：从文件中解析日志。该选项强制使用。

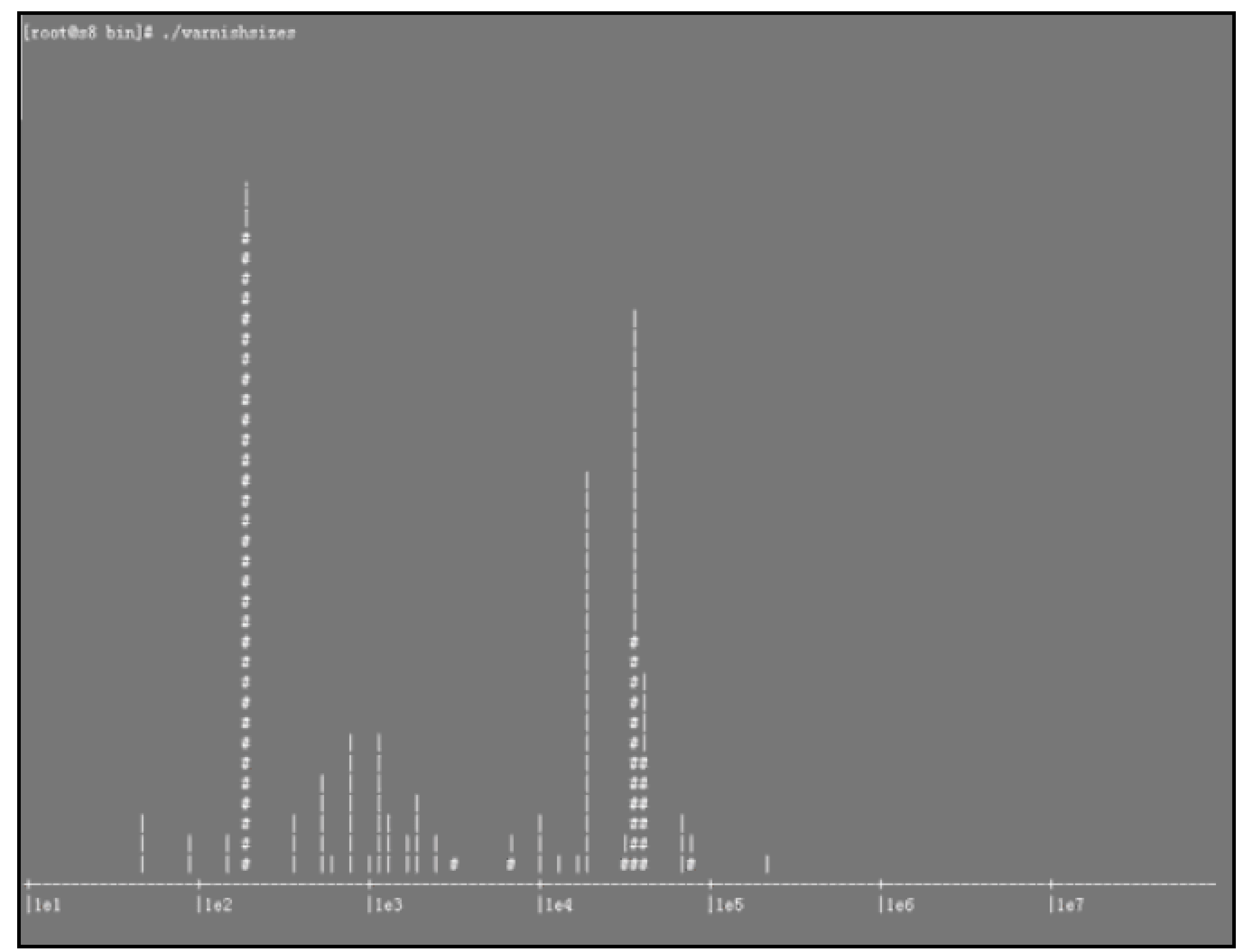
6. varnishsizes

varnishsizes 工具用于显示请求 Varnish 对象大小的柱状图。该工具通过读取 varnishd 的共享内存日志，并且以柱状图的方式显示，它会持续不断地更新，柱状图表现出最后被处理的第 N 个请求，我们注意到在图的左上角有 N 的值和纵坐标（垂直）缩放比例；横坐标（水平）方向的缩放是以对数形式呈现。我们在图中还看到有两种符号，一种是管道符（|），而另一种是哈希字符（#），它们代表着不同的意义：如果被命中（hit）那么就会标记为“|”，而没有命中的则标记为“#”。

语法格式：

```
[root@cache bin]# ./varnishsizes  -h
./varnishsizes: invalid option -- h
usage: varnishsizes [-bCcd] [-i tag] [-I regexp]
           [-k keep] [-m tag:regex] [-n varnish_name]
           [-r file] [-s skip] [-X regexp] [-x tag]
[-n varnish_name] [-V] [-w delay]
```

- -b：包含同后台服务器通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishsizes 将会视为这两者都被指定。



- -C：当使用正则表达式匹配时，忽略大小写。
- -c：包含同客户端通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishsizes 将会视为这两者都被指定。
- -d：在 varnishsizes 启动时处理旧的日志条目。通常情况下，varnishsizes 仅处理在它启动之后生成的日志。
- -I regex：只处理匹配指定正则表达式的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -i tag：处理指定 tag 的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。
- -n：该参数用于指定 varnishd 实例的名字，以便 varnishsizes 工具处理指定实例的日志。如果没有-n 参数，那么当前物理服务器的名字将会被使用。
- -r file：从指定的日志文件中读取日志条目，而不是从共享内存中读取。
- -V：显示版本号并退出。
- -w delay：该参数用于设定每一次升级间隔，默认值为 1 秒。
- -X regex：通过匹配指定的正则表达式来排除不处理的日志条目。
- -x tag：通过指定 tag 来排除不处理的日志条目。

7. varnishtest

varnishtest 工具通过使用一个命令行提供的脚本来测试 varnish HTTP 加速器。

varnishtest 工具在命令行启动时给定一个或者是多个脚本文件，它的原理在于建立一定数量的线程代表后端服务器（backend），一些线程代表客户端（client），以及一些 varnishd 进程。

语法格式：

```
[root@cache bin]#./varnishtest -h
./varnishtest: invalid option -- h
usage: varnishtest [options] file ...
-D name=val  # Define macro for use in scripts
-j jobs  # Run this many tests in parallel
-k  # Continue on test failure
-l  # Leave /tmp/vtc.* if test fails
-L  # Always leave /tmp/vtc.*
-n iterations# Run tests this many times
-q  # Quiet mode: report only failues
-t duration  # Time tests out after this long
-v  # Verbose mode: always report test log
```

下面我们看一下这些参数。

- -D name=val：该参数用于定义脚本中的宏定义。
- -j jobs：设定并行测试。
- -k：在测试中如果有测试失败的情况，不退出，而是继续测试。
- -l：如果测试失败，那么留下/tmp/vtc.*。目录的结构是这样的：

```
[root@cache ~]# tree  /tmp/vtc.17313.62841e88/
/tmp/vtc.17313.62841e88/
|-- INFO
|-- LOG
'-- v1
'--  S

1 directory, 3 files
```

- -n iterations：指定运行测试的次数。
- -L：总是留下/tmp/vtc.*。
- -q：安静模式，仅报告失败情况。
- -t duration：该属性用于指定测试时间的长度，超过这个时间就退出。
- -v：详细模式，总是报告测试日志。

这里给定的是一个脚本文件。示例如下：

```
# Start a varnish instance called "v1"
varnish v1 -arg "-b localhost:9080" -start

# Create a server thread called "s1"
server s1 {
# Receive a request
rxreq
# Send a standard response
txresp -hdr "Connection: close" -body "012345\n"
}

# Start the server thread
server s1 -start

# Create a client thread called "c1"
client c1 {
# Send a request
txreq -url "/"
# Wait for a response
rxresp
# Insist that it be a success
expect resp.status == 200
}

# Run the client
client c1 -run

# Wait for the server to die
```

```
    server s1 -wait

    # (Forcefully) Stop the varnish instance.
    varnish v1 -stop
```

测试脚本的输出：

```
#  TEST tests/b00000.vtc starting
### v1  CMD: cd ../varnishd && ./varnishd -d -d -n v1 -a :9081 -T :9001 -b
localhost:9080
### v1  opening CLI connection
#### v1  debug| NB: Storage size limited to 2GB on 32 bit architecture,\n
#### v1  debug| NB: otherwise we could run out of address space.\n
#### v1  debug| storage_file: filename: ./varnish.Shkoq5 (unlinked) size
2047 MB.\n
### v1  CLI connection fd = 3
#### v1  CLI TX| start
#### v1  debug| Using old SHMFILE\n
#### v1  debug| Notice: locking SHMFILE in core failed: Operation not
permitted\n
#### v1  debug| bind(): Address already in use\n
#### v1  debug| rolling(1)...
#### v1  debug| \n
#### v1  debug| rolling(2)...\n
#### v1  debug| Debugging mode, enter "start" to start child\n
### v1  CLI 200 <start>
##  s1  Starting server
### s1  listen on :9080 (fd 6)
##  c1  Starting client
##  c1  Waiting for client
##  s1  started on :9080
##  c1  started
### c1  connect to :9081
### c1  connected to :9081 fd is 8
#### c1  | GET / HTTP/1.1\r\n
#### c1  | \r\n
### c1  rxresp
#### s1  Accepted socket 7
### s1  rxreq
#### s1  | GET / HTTP/1.1\r\n
#### s1  | X-Varnish: 422080121\r\n
#### s1  | X-Forwarded-For: 127.0.0.1\r\n
#### s1  | Host: localhost\r\n
#### s1  | \r\n
```

```
#### s1  http[ 0] | GET
#### s1  http[ 1] | /
#### s1  http[ 2] | HTTP/1.1
#### s1  http[ 3] | X-Varnish: 422080121
#### s1  http[ 4] | X-Forwarded-For: 127.0.0.1
#### s1  http[ 5] | Host: localhost
#### s1  | HTTP/1.1 200 Ok\r\n
#### s1  | Connection: close\r\n
#### s1  | \r\n
#### s1  | 012345\n
#### s1  | \r\n
##  s1  ending
#### c1  | HTTP/1.1 200 Ok\r\n
#### c1  | Content-Length: 9\r\n
#### c1  | Date: Mon, 16 Jun 2008 22:16:55 GMT\r\n
#### c1  | X-Varnish: 422080121\r\n
#### c1  | Age: 0\r\n
#### c1  | Via: 1.1 varnish\r\n
#### c1  | Connection: keep-alive\r\n
#### c1  | \r\n
#### c1  http[ 0] | HTTP/1.1
#### c1  http[ 1] | 200
#### c1  http[ 2] | Ok
#### c1  http[ 3] | Content-Length: 9
#### c1  http[ 4] | Date: Mon, 16 Jun 2008 22:16:55 GMT
#### c1  http[ 5] | X-Varnish: 422080121
#### c1  http[ 6] | Age: 0
#### c1  http[ 7] | Via: 1.1 varnish
#### c1  http[ 8] | Connection: keep-alive
#### c1  EXPECT resp.status (200) == 200 (200) match
##  c1  ending
##  s1  Waiting for server
#### v1  CLI TX| stop
### v1  CLI 200 <stop>
#  TEST tests/b00000.vtc completed

If instead of 200 we had expected 201 with the line::

expect resp.status == 201

The output would have ended with::

#### c1  http[ 0] | HTTP/1.1
```

```
#### c1  http[ 1] | 200
#### c1  http[ 2] | Ok
#### c1  http[ 3] | Content-Length: 9
#### c1  http[ 4] | Date: Mon, 16 Jun 2008 22:26:35 GMT
#### c1  http[ 5] | X-Varnish: 648043653 648043652
#### c1  http[ 6] | Age: 6
#### c1  http[ 7] | Via: 1.1 varnish
#### c1  http[ 8] | Connection: keep-alive
---- c1  EXPECT resp.status (200) == 201 (201) failed
```

这个例子是 man 文档中的例子，对于该命令，只作一下了解而已，在实际的使用中，我测试了几个例子并不理想。

8. varnishtop

varnishtop 工具通过读取 varnishd 共享内存日志，并且以持续不断的方式列出了当前最常发生的日志条目。通过适当使用-I、-i、-X 和-x 选项来过滤显示请求文档、客户端、用户代理或者是在日志中记录的其他任何信息。

例如：

```
[root@s8 bin]# ./varnishtop
list  length  2515  s8.xx.cn

  2956.78 VCL_return deliver
  1520.28 TxProtocol HTTP/1.1
  1478.39 VCL_call   deliver
  1478.39 TxHeader   Via: 1.1 varnish
  1478.39 TxHeader   Connection: close
  1478.39 SessionClose   Connection: close
  1478.39 VCL call   recv
  1478.39 VCL_return lookup
  1478.39 VCL_call   hash
  1478.39 VCL_return hash
  1478.39 RxRequest  GET
  1478.39 RxProtocol HTTP/1.0
  1478.39 RxHeader   Host: 127.0.0.1:5555
  1478.39 RxHeader   Connection: close
  1436.50 VCL_call   hit
  1367.64 RxHeader   Accept-Encoding: gzip, deflate
  1270.94 RxHeader   Accept-Language: zh-cn
  1056.45 RxHeader   Accept: */*
  1013.41 TxHeader   Server: Apache/2.2.3 (Unix)
   969.50 TxStatus   200
   969.50 TxResponse OK
   493.75 TxHeader   Content-Type: text/html
   464.98 TxStatus   304
```

```
      464.98 TxResponse Not Modified
      464.98 Length 0
      411.03 RxHeader   User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.1; SV1)
      321.16 TxHeader   Content-Type: image/gif
      276.95 TxHeader   Date: Wed, 22 Jun 2011 07:07:01 GMT
      272.31 RxHeaderReferer: http://happy.xx.cn/346/c/201106/22/n3381800,
 9.shtml
      266.70 TxHeader   Date: Wed, 22 Jun 2011 07:07:00 GMT
      243.97 TxHeader   Date: Wed, 22 Jun 2011 07:06:57 GMT
      237.26 TxHeader   Date: Wed, 22 Jun 2011 07:06:58 GMT
      210.94 TxHeader   Date: Wed, 22 Jun 2011 07:06:56 GMT
      200.57 TxHeader   Date: Wed, 22 Jun 2011 07:06:59 GMT
```

语法格式：

```
[root@cache bin]# ./varnishtop -h
./varnishtop: invalid option -- h
usage: varnishtop [-bCcd] [-i tag] [-I regexp] [-k keep]
          [-m tag:regex] [-n varnish name] [-r file]
          [-s skip] [-X regexp] [-x tag] [-1fV] [-n varnish_name]
```

下面我们看一下这些参数。

- -1：如果使用该参数，那么 varnishtop 仅将当前状态显示到屏幕就立刻退出，而不是持续不断地更新显示。使用该参数包含参数-d。
- -b：包含同后台服务器通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishtop 将会视为这两者都被指定。
- -C：当使用正则表达式匹配时，忽略大小写。
- -c：包含同客户端通信的日志条目。如果既没有指定-b 也没有指定-c，那么 varnishtop 将会视为这两者都被指定。
- -d：在 varnishlog 启动时处理旧的日志条目。通常情况下，varnishtop 仅处理在它启动之后生成的日志。
- -f：仅对每一个日志条目的第一个字段分组和排序。
- -I regex：只处理匹配指定正则表达式的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。例如：

```
[root@cache bin]# ./varnishtop  -i RxHeader -C -I ^User-Agent
list length 53
62.77 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1)
55.90 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1; GTB6.4; SE 2.X MetaSr 1.0)
28.78 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; I
27.80 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT
```

```
6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
    12.96 RxHeader  User-Agent: Googlebot-News
    10.97 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; QQPinyin 689)
    10.95 RxHeader  User-Agent: Mozilla/4.0
     7.00 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727)
     5.98 RxHeaderUser-Agent: HuaweiSymantecSpider/1.0+DSE-support@huaweisymantec.com+(compatible; MSIE 7.0; Windows NT 5.1; Tride
     5.96 RxHeader  User-Agent: Mozilla/5.0 ( Windows NT 5.1 ) AppleWebKit/534.24 (KHTML, like Gecko) Chrome/11.0.696.71 Safari/534.24
     5.00 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
     5.00 RxHeader  User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.10 (KHTML, like Gecko) Chrome/8.0.55
     4.98 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727)
     4.98 RxHeader  User-Agent: Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; QQDownload 1.7; .NET CLR 2.0.50727;
     1.99 RxHeader  User-agent: Baiduspider-cpro
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; User-agent: Mozilla/4.0 (compatible; MSIE 6.
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
     1.99 RxHeaderUser-Agent:Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+5.1;+.NET+CLR+2.0.50727)
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
     1.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; GTB6.6)
     1.00 RxHeaderUser-Agent:Sogouwebspider/4.0(+http://www.sogou.com/docs/help/webmasters.htm#07)
     1.00 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; GTB0.0; .NET CLR 2.0.50727; .NET CLR
     1.00 RxHeader  User-Agent: Microsoft URL Control - 6.00.8862
     1.00 RxHeader  User-Agent: Mozilla/5.0 (compatible; YodaoBot-News/1.0; http://www.youdao.com/help/webmaster/spider/; )
     1.00 RxHeaderUser-Agent:AppEngine-Google;(+http://code.google.com/appengine; appid: 4355fg)
```

```
    1.00 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; CNCDialer; .NET CLR 1.1.4322; InfoPath.1; .N
    1.00 RxHeader  User-Agent: Nokia5230/UCWEB7.2.2.51/50/800
    1.00 RxHeader  User-Agent: Nokia5230/UCWEB7.6.0.75/50/999
    1.00 RxHeaderUser-Agent:Sosospider+(+http://help.soso.com/webspider.htm)
    0.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; Media Center PC 5.0; .
    0.99 RxHeader  User-Agent: Mozilla/5.0 ( compatible; MJ12bot/v1.3.3; http://www.majestic12.co.uk/bot.php?+)
    0.99 RxHeader  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; GTB6.5; TheWorld)

   … //省略部分
```

- -i tag：处理指定 tag 的日志条目。如果既没有指定-I 也没有指定-i，那么将会处理所有的日志条目。例如：

```
[root@cache bin]# ./varnishtop -i RxURL

list  length 129
16.96 RxURL /images/img index/1/net bn.gif
11.97 RxURL /images/img index/1/baidu.gif
 3.99 RxURL /images/index images/new/daoying.png
 3.00 RxURL /images/index images/new/006.png
 3.00 RxURL /images/index_images/new/007.png
 3.00 RxURL /images/index_images/new/004.png
 2.99 RxURL /images/index_images/new/new.gif
 2.99 RxURL /images/index images/new/wsdb.gif
 2.99 RxURL /images/index images/2010/transparent.gif
 2.99 RxURL /images/index images/new/sjdy.gif
 2.99 RxURL /images/index images/new/003.png
 2.99 RxURL /lib/css/news/index/guid.css
 2.99 RxURL /lib/js/news/index/huandeng.js
 2.99 RxURL /images/index_images/new/index_bg_01.gif
 2.99 RxURL /images/index_images/new/sy.gif
 2.99 RxURL /images/index images/new/sc.gif
 2.99 RxURL /images/index images/new/index bg 18.gif
 2.99 RxURL /images/index images/new/002.png
 2.99 RxURL /images/index_images/new/001.png
 2.99 RxURL /images/index_images/new/ico_139_0.gif
 2.99 RxURL /images/index_images/new/kaixin001_0.gif
 2.99 RxURL /images/index_images/new/ico_qw_0.gif
 2.99 RxURL /images/index_images/new/ico_rss_0.gif
```

```
 2.99 RxURL /images/images_home/3/jhxw_title_pic_004

… //省略部分
```

- -n：该参数用于指定 varnishd 实例的名字，以便 varnishtop 工具处理指定实例的日志。如果没有-n 参数，那么当前物理服务器的名字将会被使用。
- -r file：从文件中读取日志条目，而不是从共享内存。
- -V：显示版本号并退出。
- -X regex：通过匹配指定的正则表达式来排除不处理的日志条目。
- -x tag：通过指定 tag 来排除不处理的日志条目。

57.12 手动清除缓存

我们知道，要想提高缓存的命中率，就得增加缓存对象的 TTL。然而，在增大缓存的同时也会存在很多的无用数据，或者是需要更新页面，等等，都需要删除缓存对象。

在前面讲述命令时，已经了解到手动清除缓存的操作，在这里我们更深入地了解一下。手动清除缓存，有时候非常有用。举一个比较常见的例子，无论是哪家网站都删过稿子（或者叫网站最终页），如果一篇稿子在后台或者是在存储上已经删除，而客户端还能从你运维的网站访问（而你设定的缓存时间为 84600 秒），此时必须不手动清除缓存。

2.1.5 版的命令

```
[root@s8 bin]# ./varnishadm  -T 127.0.0.1:2000
200 191
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.9-5.ELsmp,i686,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.

… //省略
purge.url <regexp>
purge <field> <operator> <arg> [&& <field> <oper> <arg>]...
purge.list
```

3.0.0 稳定版的命令

```
[root@mail bin]# ./varnishadm -T 127.0.0.1:2000
CLI connected to 127.0.0.1:2000
200
-----------------------------
Varnish Cache CLI 1.0
```

```
------------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.

… //省略
panic.show
panic.clear
storage.list
ban.url <regexp>
ban <field> <operator> <arg> [&& <field> <oper> <arg>]...
ban.list
```

这些我们在前面都介绍过，而且也讲述过各个命令的用法。需要明确的一点是，在 2.x 中使用的是 purge 方式，而在 3.x 中使用了 ban 方式。

57.12.1 基于命令行方式清除 Varnish 缓存

不同版本的 Varnish 删除命令的写法和命令本身内在的原理不尽相同，但是它给我们的表现却是功能相同，因此我们在不同的版本中可以按照同一个功能来使用：

- url.purge、purge.url 和 ban.url 这三个命令的功能相同，并且支持正则表达式；
- ban、purge 这两个命令的功能相同，同样支持正则表达式；
- ban.list、purge.list 这两个命令的功能相同。

基于命令行清除缓存有三种方法。

基于命令行的 varnishadm 命令清除缓存的方式在前面介绍命令时已有说明，通过 telnet 到管理端口来实现清除功能的方法和 varnishadm 命令相同。需要说明的是基于 HTTP 的方法，它解决了跨平台、跨机器，即不用进入 Varnish 所在的机器进行操作，而且还提供了限制 IP 操作，这种方式虽然好用，但是需要配置 Varnish 的 VCL，因此，在实际的生产环境中我并没有使用，在这里只介绍一下，生产环境下不建议使用。

1. 第一种方法：telnet 到管理端口

例如：

```
[root@mail ~]# telnet 127.0.0.1 2000
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1) .
Escape character is '^]'.
200 193
------------------------------
Varnish Cache CLI 1.0
------------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit
```

```
Type 'help' for command list.
Type 'quit' to close CLI session.
```

2. 第二种方法：使用 varnishadm 命令

例如：

```
[root@mail bin]# ./varnishadm -T 127.0.0.1:2000
CLI connected to 127.0.0.1:2000
200
-----------------------------
Varnish Cache CLI 1.0
-----------------------------
Linux,2.6.25,i686,-smalloc,-smalloc,-hcritbit

Type 'help' for command list.
Type 'quit' to close CLI session.
```

前两种方法我们基本都已了解，下面我们来看一下第三种方法。

3. 第三种方法：telnet 到 80 端口

理解 http 的 purge 和 ban 之间的区别很重要，purge 方式是通过立刻将缓存对象的生存期设置为 0，这就意味着缓存过期；而 ban 则是在 Varnish 内部通过添加一个表示方法将其列入 ban 的列表而不再提供访问，被列入列表的对象可以使用 ban.list 命令查看，但是如果列在列表中的对象超出生存期，那么对象将不再列出。

purge 方式的优势在于清除的对象立刻清除，被清除的对象在 Varnish 中不再占用额外的资源；而它的劣势是只能清除单个对象，就是说对于同一个被缓存的访问对象（例如，html 文件或是 jpg 文件等等）的变体（由 Vary 产生）不会被清除，这就是说，如果清除压缩和非压缩的两个版本，那么我们不得不清除两次。

ban 方式的优势在于它使用了添加表示方法（即使用列表禁用方式），因此我们可以禁止 URL 空间的一部分，因此就可以实现对某个虚拟主机或者是其他的 location 进行“查封”；

但是 ban 方式同样有它的劣势，每次执行 ban 都要核对每一个被缓存的对象，因此它比 purge 使用更多的资源，如果添加了很多清除，它们将消耗大量的内存。

这种方式在 Varnish 的 2.1.和 3.0 的 VCL 中配置有所不同。首先来看一下在 2.1.5 中的配置：

purge 方式：

HTTP purge 除了使用的方法是 purge 之外，它类似于 HTTP GET 请求，实际上我们可以在任何时候调用该方法，但是大多数人提到它就是做清除工作。SQUID 支持同样的机制。为了支持清除操作，在 Varnish 中添加一些代码。

```
acl purge {
"localhost";
"192.168.4.0"/24;
}
```

```
sub vcl_recv {

if (req.request == "PURGE") {
if (!client.ip ~ purge) {
error 405 "Not allowed.";
}
return (lookup);
}
}

sub vcl_hit {
if (req.request == "PURGE") {
# Note that setting ttl to 0 is magical.
# the object is zapped from cache.
set obj.ttl = 0s;
error 200 "Purged.";
}
}

sub vcl_miss {
if (req.request == "PURGE") {

error 404 "Not in cache.";
}
}
```

正如我们看到的，在这里我们使用了新的 VCL 子程序：vcl_hit 和 vcl_miss，当我们调用 lookup 的时候，Varnish 将试图在它的缓存中查找相应的对象，这样就会有两种可能，一种是找到指定的对象，另一种是没找到指定的对象，那么就会根据具体的情况来调用相应的值函数 vcl_hit 或 vcl_miss。如果是 vcl_hit 情况，也就说，对象在缓存中存储有效，就可以对其设置 TTL 了。

因此，要使得 xx.com 的首页无效，那么可以执行：

```
PURGE / HTTP/1.0
Host: xx.com
```

执行该命令后，Varnish 将会使得其首页无效，然而，可能在缓存中还有相同 URL 的变体副本，这种方法只能清除匹配的变体副本，因此要想清除相同网页的 gzip 方式压缩的变体副本，那么需要执行以下的方式：

```
PURGE / HTTP/1.0
Host: xx.com
```

```
Accept-Encoding: gzip
```

在 2.1.5 中是这样，但是在 3.0.0 中有了改变，通过：

```
PURGE / HTTP/1.0
Host: xx.com
```

就可以清除掉所有的主页变体，而不用再清除其他的变体方式了。

ban 方式：

使内容作废的另一种方式就是 ban，ban 的操作可以想象为过滤的排序，我们可以“查封”从缓存中提供的内容，是基于元数据。对于 ban 支持的操作在 Varnish 提供的命令行，同样能够做到，例如，“查封”xx.com 的所有 png 文件，那么可以在管理接口的命令行中执行以下命令：

```
purge req.http.host == "vg.no" && req.http.url ~ "\.png$"
```

ban 方式会在当期望的对象在缓存中命中（hit），但又是在交付（deliver）之前进行检查，命中的缓存对象会被新的 ban 检查，因此如果有很多缓存对象，而且 TTL 也较长，那么不得不考虑性能，大量的 ban 操作的签字会影响 Varnish 的性能。

同样，ban 也可以通过 HTTP 方式来操作，在 VCL 中添加以下代码：

```
sub vcl recv {
if (req.request == "BAN") {
# Same ACL check as above:
if (!client.ip ~ purge) {
error 405 "Not allowed.";
}
purge ("req.http.host == " req.http.host
  "&& req.url == " req.url) ;

# Throw a synthetic page so the
# request wont go to the backend.
error 200 "Ban added";
}
}
```

这段 VCL 代码启用了通过 HTTP ban 方法执行“查封”操作，实际上真正执行的是 purge 命令，即黑体字部分。

下面我们看一下在 3.0.0 中的配置。

purge 方式

```
# Who is allowed to purge....
acl local {
"localhost";
"192.168.3.0"/24; /* and everyone on the local network */
! "192.168.1.23"; /* except for the dialin router */
}

sub vcl_recv {
```

```
  if (req.request == "PURGE") {
if (client.ip ~ local) {
   return (lookup);
}
  }
}

sub vcl_hit {
   if (req.request == "PURGE") {
 set obj.ttl = 0s;
 error 200 "Purged.";
}
}

sub vcl_miss {
  if (req.request == "PURGE") {
error 404 "Not in cache.";
  }
}
```

ban 方式

```
sub vcl_recv {
if (req.request == "BAN") {
# Same ACL check as above:
if (!client.ip ~ purge) {
error 405 "Not allowed.";
}
ban ("req.http.host == " + req.http.host +
  "&& req.url == " + req.url);

# Throw a synthetic page so the
# request won't go to the backend.
error 200 "Ban added";
}
}
```

telnet 到 80 端口清除缓存对象

这里的端口 8421 是 Varnish 监听的 Web 端口，可以通过 http 协议直接访问。下面我们通过 telnet 的方式来清除/images/t.php 的缓存：

```
[root@mail cm]# telnet www.xx.com 8421
Trying 192.168.18.46...
Connected to app1.xx.com (192.168.18.46).
Escape character is '^]'.
```

```
PURGE /images/t.php HTTP/1.1
Host:www.xx.com

HTTP/1.1 200 Purged.
Server: Varnish
Retry-After: 0
Content-Type: text/html; charset=utf-8
Content-Length: 382
Date: Sun, 14 Aug 2011 11:03:55 GMT
X-Varnish: 815779858
Age: 0
Via: 1.1 varnish
Connection: close


<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
<title>200 Purged.</title>
  </head>
  <body>
<h1>Error 200 Purged.</h1>
<p>Purged.</p>
<h3>Guru Meditation:</h3>
<p>XID: 815779858</p>
<hr>
<p>Varnish cache server</p>
  </body>
</html>
Connection closed by foreign host.
```

注意黑体字<html> ... </html>之间的内容为返回的网页，我们已经成功地清除了/images/t.php 的缓存。

如果出现以下内容，那么则说明在缓存中没有找到相应的条目：

```
[root@mail varnish-3.0.0-beta1]# telnet www.xx.com 8421
Trying 192.168.18.46...
Connected to app1.xx.com (192.168.18.46).
Escape character is '^]'.
PURGE /images/t.php HTTP/1.1
Host:www.xx.com
```

```
HTTP/1.1 404 Not in cache.
Server: Varnish
Retry-After: 0
Content-Type: text/html; charset=utf-8
Content-Length: 401
Date: Sun, 14 Aug 2011 23:49:20 GMT
X-Varnish: 1419289416
Age: 0
Via: 1.1 varnish
Connection: close


<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
<title>404 Not in cache.</title>
  </head>
  <body>
<h1>Error 404 Not in cache.</h1>
<p>Not in cache.</p>
<h3>Guru Meditation:</h3>
<p>XID: 1419289416</p>
<hr>
<p>Varnish cache server</p>
  </body>
</html>
```

如果出现以下内容，那么说明执行 telnet 命令所在机器的 IP 被列为不允许用作清除操作的 IP：

```
[root@mfsmaster ~]# telnet www.xx.com 8421
Trying 192.168.18.46...
Connected to www.xx.com (192.168.18.46).
Escape character is '^]'.
PURGE /images/t.php HTTP/1.1
Host:www.xx.com

HTTP/1.1 405 Not allowed.
Server: Varnish
Retry-After: 0
Content-Type: text/html; charset=utf-8
Content-Length: 397
Date: Mon, 15 Aug 2011 00:00:42 GMT
```

```
X-Varnish: 228534303
Age: 0
Via: 1.1 varnish
Connection: close


<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
<title>405 Not allowed.</title>
  </head>
  <body>
<h1>Error 405 Not allowed.</h1>
<p>Not allowed.</p>
<h3>Guru Meditation:</h3>
<p>XID: 228534303</p>
<hr>
<p>Varnish cache server</p>
  </body>
</html>
Connection closed by foreign host.
```

57.12.2 基于应用程序方式清除 Varnish 缓存

对于使用应用程序清除 Varnish 缓存无非也就是将命令行方式进行某种语言的封装，可以使用 PHP，也可使用 Perl 或者是 Python 来实现，在这里我们看一下 PHP 的方式。但是，我更喜欢使用 varnishadm 命令来操作（这样我们才更有用！！！）。

1. Web 系统架构

我们看一个完整的例子，在这个例子中分为三级，即“Nginx——Varnish——Apache”，我们需要配置 Nginx，而且要注意下面配置中的黑体字部分：

```
server {

  listen   80;
  server_name  www.xx.cn;
  #  ssi on ;
  index index.shtml index.htm index.php;

  location / {
proxy_set_header   Host $host;
proxy_set_header   X-Real-IP$remote_addr;
proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
```

```
proxy_passhttp://127.0.0.1:8421;

  }
```

然后就是 Varnish 的配置文件：

```
backend ICBC {
   .host = "192.168.18.60";
   .port = "80";
}

acl purge {
   "localhost";
   "127.0.0.1";
   "192.168.18.0"/24;
}

sub vcl_recv {
   if (req.request == "PURGE") {
   if (!client.ip ~ purge) {
   error 405 "Not allowed.";
   }
   return (lookup);
   }

   if (req.http.host ~ "^www.xx.com") {
   set req.backend = ICBC;
   if (req.request != "GET" && req.request != "HEAD") {
   return (pipe);
   }
   else {
   return (lookup);
   }
   }
   else {
   error 404 " Sorry,domain name wrong! ";
   return (lookup);
   }
}

sub vcl_hit {
   if (req.request == "PURGE") {
   set obj.ttl = 0s;
   error 200 "Purged.";
   }
```

```
}

sub vcl_miss {
  if (req.request == "PURGE") {
  error 404 " NO CACHED PAGE FOUND! ";
  }
}

sub vcl_fetch {
  if (req.request == "GET" && req.url ~ "\.(txt|js|php|css|html)$") {
  set beresp.ttl = 1800s;
  }
  else {
  set beresp.ttl = 7d;
  }
}
```

最后是用于清除缓存的 PHP 文件：

```
[root@c60 web60]# cat purge.php
<?php
function varnish_purge($ip, $host, $url)
{
$errstr = 'You are wrong! ';
$errno = '';
$fp = fsockopen ($ip, 5555, $errno, $errstr, 10);
if (!$fp)
{
 return false;
}
else
{
$out = "PURGE {$url} HTTP/1.1\r\n";
$out .= "Host:{$host}\r\n";
$out .= "Connection: close\r\n\r\n";
fputs ($fp, $out);
$out = fgets($fp , 4096);
fclose ($fp);
return true;
}
}

varnish purge("192.168.18.46", "www.xx.com", $ GET['f']);
echo "Purged: ";
echo $_GET['f'];
```

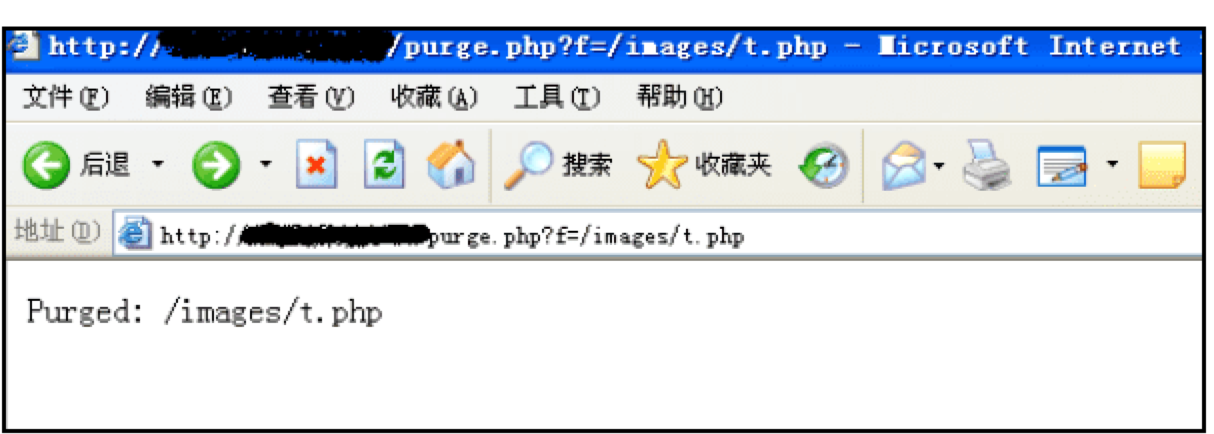
```
?>
```

在这个 PHP 文件中，定义了一个 Varnish 缓存的函数，需要三个参数，$ip 为 Varnish 服务器的 IP 地址，在连接服务器时使用，$host 为被清除或者叫刷新页面网站的域名，而$url 为要刷新的不含域名部分的 URL 地址。在该函数中还有一点，那就是使用了$_GET['f']变量，用于从浏览器传入被清除的网页地址。

2. 测试访问

访问 http://192.168.18.60/purge.php?f=/images/t.php 或者 http://192.168.18.46/purge.php?f=/images/t.php 都可以，但是不要以这种方式访问 http://www.xx.com/purge.php?f=/images/t.php，因为会被缓存。实际上，我们使用第一种方式更好，即直接访问后端的 Apache 服务器。当然在实际的环境中我们也未必缓存 PHP 文件，但使用这种清除方式就是能够实现只要符合 Varnish 中配置的清除 IP，而且还有 PHP 环境，那么就可以实现清除操作。

这是返回的页面，如果你对 PHP 感兴趣，那么可以更好地完善这个 PHP 文件，以上这个文件的内容仅是在清除 SQUID 缓存基础上修改而成，因此还有许多地方可以做，我对编程不感兴趣，因此，够用就行。



同时我们可以监控访问日志，对于 Nginx 就不用监控了，这个访问就没有走 Nginx，我们使用的是第一种方法：http://192.168.18.60/purge.php?f=/images/t.php。

下面是 Varnish 的访问日志：

```
192.168.18.60 - - [14/Aug/2011:21:16:02 +0800] "PURGE /images/t.php HTTP/1.1" 200 383 "-" "-"
```

下面是 Apache 的日志

```
192.168.3.248 --[14/Aug/2011:21:05:08+0800]"GET /purge.php?f=/images/t.php HTTP/1.1" 200 21
```

注意黑体字部分，访问在不同的阶段，会有不同的 HTTP 方式。

看下面一种情况，由于我们在 Varnish 的 VCL 文件中只配置了 www.xx.com，而 app1.xx.com 与该域名解析到了同一个 IP，按理说执行清除是没有问题的，可是由于配置的约束，只能得到以下响应。



可以看一下访问日志，以下是 Nginx 的访问日志：

```
192.168.3.248 --[15/Aug/2011:08:27:24+0800]"GET /images/purge.php?f=/
images/t.php HTTP/1.1" 404 434 "-" "Mozilla/5.0 (Windows; U; Windows NT
5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

Varnish 的访问日志：

```
127.0.0.1 --[15/Aug/2011:08:27:24+0800]"GET http://app1.xx.com/images/
purge.php?f=/images/t.php HTTP/1.0" 404 434 "-" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 GTB7.1"
```

它们的状态均为 404，Apache 的日志就不用看了，访问还没到这一步。

57.13 VCL 语言

VCL 是一个小领域的特定语言，它所描述的是为 Varnish HTTP 加速定义请求处理和文档缓存策略。当一个新的配置被载入时，varnishd 管理进程（management process）会将 VCL 代码转换为 C，将其编译为一个共享对象，然后被动态地链接到服务器进程（server process）。

VCL 的语法十分简单，与 C 和 Perl 语言十分相似。在定义中，每个块之间使用花括号“{}”分隔，语句间是以分号“；”分隔。对于注释语句，可以根据自己的喜好使用 C、C++ 或者 Perl 的注释格式都可以，不过我个人还是喜欢用“#”号注释。

另外，VCL 除了支持类 C 的赋值运算符（=），比较运算符（==）及布尔运算符（!、&&和||），还支持正则表达式，ACL 使用“~”作为匹配操作符。

VCL 不像 C 和 Perl，在字符串中的反斜线（\）字符出现在 VCL 中没有特定的意义，只是用来匹配 url，因此，可以在正则表达式中自由使用，而不必采用双反斜线。

字符串相连接只是通过将它们一个接一个放置在一起，它们之间并没有任何操作符。

其实 VCL 只是一个简单的配置语言而已，并不像真正的编程语言，它有语法、变量、关键字、函数等，但是没有自定义变量，VCL 有 if 测试语句而没有循环语句，等等，因此，它并不是真正意义上的编程语言。

57.13.1 默认配置文件

首先我们看一下默认的配置文件：

```
[root@cache etc]# more default.vcl
# This is a basic VCL configuration file for varnish.  See the vcl(7)
# man page for details on VCL syntax and semantics.
```

```
#
# Default backend definition.  Set this to point to your content
# server.
#
# backend default {
# .host = "127.0.0.1";
# .port = "8080";
# }
#
# Below is a commented-out copy of the default VCL logic.  If you
# redefine any of these subroutines, the built-in logic will be
# appended to your code.
# sub vcl recv {
# if (req.restarts == 0) {
#   if (req.http.x-forwarded-for) {
#   set req.http.X-Forwarded-For =
#   req.http.X-Forwarded-For + ", " + client.ip;
#   } else {
#   set req.http.X-Forwarded-For = client.ip;
#   }
# }
# if (req.request != "GET" &&
#   req.request != "HEAD" &&
#   req.request != "PUT" &&
#   req.request != "POST" &&
#   req.request != "TRACE" &&
#   req.request != "OPTIONS" &&
#   req.request != "DELETE") {
# /* Non-RFC2616 or CONNECT which is weird. */
# return (pipe);
# }
# if (req.request != "GET" && req.request != "HEAD") {
# /* We only deal with GET and HEAD by default */
# return (pass);
# }
# if (req.http.Authorization || req.http.Cookie) {
# /* Not cacheable by default */
# return (pass);
# }
# return (lookup);
# }
#
# sub vcl_pipe {
```

```
# # Note that only the first request to the backend will have
# # X-Forwarded-For set.  If you use X-Forwarded-For and want to
# # have it set for all requests, make sure to have:
# # set bereq.http.connection = "close";
# # here.  It is not set by default as it might break some broken web
# # applications, like IIS with NTLM authentication.
# return  (pipe) ;
# }
#
# sub vcl_pass {
# return  (pass) ;
# }
#
# sub vcl hash {
# hash_data (req.url) ;
# if  (req.http.host)  {
# hash_data (req.http.host) ;
# } else {
# hash data (server.ip) ;
# }
# return  (hash) ;
# }
#
# sub vcl_hit {
# return  (deliver) ;
# }
#
# sub vcl miss {
# return  (fetch) ;
# }
#
# sub vcl_fetch {
# if  (beresp.ttl <= 0s ||
# beresp.http.Set-Cookie ||
# beresp.http.Vary == "*")  {
#   /*
#* Mark as "Hit-For-Pass" for the next 2 minutes
#*/
#   set beresp.ttl = 120 s;
#   return  (hit_for_pass) ;
# }
# return  (deliver) ;
# }
```

```
#
# sub vcl_deliver {
# return (deliver);
# }
#
# sub vcl error {
# set obj.http.Content-Type = "text/html; charset=utf-8";
# set obj.http.Retry-After = "5";
# synthetic {"
# <?xml version="1.0" encoding="utf-8"?>
# <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
#  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
# <html>
#   <head>
# <title>"} + obj.status + " " + obj.response + {"</title>
#   </head>
#   <body>
# <h1>Error "} + obj.status + " " + obj.response + {"</h1>
# <p>"} + obj.response + {"</p>
# <h3>Guru Meditation:</h3>
# <p>XID: "} + req.xid + {"</p>
# <hr>
# <p>Varnish cache server</p>
#   </body>
# </html>
# "};
# return (deliver);
# }
```

在这个配置文件中有 if 语句，还有 sub 子函数及相关的变量设定和操作符，十分像一种高级语言。在下面的内容中我们将会对该文件进行分析，需要注意的一点是，该文件中所有的内容都被注释掉。

57.13.2　操作符

下列操作符在 VCL 中有效。

- =：赋值运算符。
- ==：比较运算符。
- ~：匹配运算符。可以用在正则表达式或者是 ACL。
- !：否定运算符。
- &&：逻辑运算符“与”。
- ||：逻辑运算符“或”。

下面是处理操作的优先规则，顺序是由高到低：

原子操作

'true'，'false'，常量

函数调用

变量

'（'表达式'）'

乘法/除法操作

INT * INT

INT / INT

DURATION * REAL

加法/减法操作

STRING + STRING

INT +/- INT

TIME +/- DURATION

TIME - TIME

DURATION +/- DURATION

比较运算操作

'=='，'!='，'<'，'>'，'~'和'!~'

字符串存在性检查（->BOOL）

逻辑运算操作

逻辑“非”：'!'

逻辑“与”：'&&'

逻辑“或”：'||'

> 需要说明的一点是：在 VCL 中我们有两种时间类型，即 TIME 和 DURATION，在 HTTP 内容中有着非同寻常的作用，既可以保持一个相对的时间（即 age），也可使用绝对时间（即 Expire）。显而易见，TIME 加上 DURATION，其结果是 TIME 类型。同样，TIME 与 TIME 是不可相加的，但是 TIME 与 TIME 却是可以相减的，其相减的结果是 DURATION。

57.13.3 数据结构

在 VCL 中存在着 request、response 和 object，它们是三种重要的数据结构：request 来自于客户端；response 来自于后台服务器；object 则是存储在缓存中的对象。

在 VCL 中我们要了解下面的三种结构。

- req：请求对象。当 Varnish 收到请求的时候，那么 req 对象就会被建立并且被存储。在 vcl_recv 中做的大多数工作就是 req 对象的设置。
- beresp：后端服务器的响应对象。它包含从后台服务器进来的对象头。在 vcl_fetch 中做的大多数工作就是 beresp 对象的设置。

- Obj：缓存对象，驻留在内存中的对象几乎都是只读的。obj.ttl 可写，其他的为只读。

我们来看下面的三个例子。

例 1：控制请求头

我们看一下这个例子，在这个例子中，我们想移除 cookie 信息，所有匹配的 url 都将移除其 cookie 信息，在这里我们针对静态图片设置：

```
sub vcl_recv {
  if (req.url ~ "^/images") {
unset req.http.cookie;
  }
}
```

通过这个配置，现在当请求被处理传到后台服务器时将不再有 cookie 头。

我们看一下 if 语句段，如果请求的 URL 与该正则表达式匹配，那么获取该对象，并且如果匹配，那么清除 cookie 请求头。

例 2：控制响应对象

在这个例子中，如果匹配 URL 中的某一个标准，那么我们覆盖掉了从后台服务器进入对象的 TTL，并且取消了 cookie 设置：

```
sub vcl_fetch {
   if (req.url ~ "\.(png|gif|jpg)$") {
 unset beresp.http.set-cookie;
 set beresp.ttl = 1h;
  }
}
```

例 3：ACL

在下面的例子中，我们通过关键字 acl 建立了访问控制列表，然后通过匹配操作符来匹配 IP 地址：

```
# Who is allowed to purge....
acl local {
"localhost";
"192.168.1.0"/24; /* and everyone on the local network */
! "192.168.1.23"; /* except for the dialin router */
}

sub vcl recv {
  if (req.request == "PURGE") {
if (client.ip ~ local) {
   return(lookup);
}
  }
}

sub vcl_hit {
```

```
    if (req.request == "PURGE") {
  set obj.ttl = 0s;
  error 200 "Purged.";
 }
 }

 sub vcl miss {
   if (req.request == "PURGE") {
 error 404 "Not in cache.";
   }
 }
```

关键字 return（action）用来终结子程序，它可以是以下的某一种：

- deliver
- error
- fetch
- hash
- lookup
- pass
- pipe
- restart

对于每个函数都是调用 return（）来终结，而对于不同的函数 return（）可选择的 action 不相同。有关这些 action 的功能将会在子程序部分解释。

57.13.4 变量

尽管子程序没有参数，然而对于必要信息，子进程必须通过全局变量来处理。以下是 VCL 中可用的变量。

以下变量总是可用：

now：当前的时间，但是采用的是纪元开始的秒。

以下变量可以在后端声明：

- .host：该变量用于指定后台服务器的 IP 地址或者是主机名。
- .port：用于指定后台服务器的服务名称，或是端口号。

以下变量可用在处理请求中：

- client.ip：该变量表示客户端 IP 地址。
- server.hostname：该变量表示服务器的名字。
- server.identity：该变量的值代表着 server 的身份，varnishd 在命令行通过-i 参数设置的，如果在命令中没有设置-i 参数的值，那就没有向 varnishd 传递 varnishd 实例名字。
- server.ip：该变量用于设置服务器端 IP 地址，用于客户端进行套接字连接。

- server.port：该变量用于设置服务器端 TCP/IP 套接字监听的端口，用于客户端进行套接字连接。
- req.request：用于设定请求的类型（例如“GET”，“HEAD”）。
- req.url：指定客户端请求的 URL。
- req.proto：该变量用于指定客户端使用的 HTTP 协议版本。
- req.backend：使用哪个后端服务器为该请求提供服务。
- req.backend.healthy：该变量用于表示后端服务器是否健康。这依赖于一个设置在后台服务器的活动探测器。
- req.http.header：相应的 HTTP 头。
- req.http.host：请求的主机名称。看下面的一个例子：

```
if (req.http.host ~ "(?i)^(www.)?varnish-?software.com") {
  set req.http.host = "varnish-software.com";
}
```

有的站点可能会有多个主机名，例如在这个例子中，它可以包含 www.varnish-software.com、varnish-software.com 或者 varnishsoftware.com，那么我们的访问者即客户端浏览器，将可能通过 http://www.varnish-software.com/、http://varnish-software.com/ 或 http://varnishsoftware.com/访问，这样就会造成一个页面的多份复制，就是访问的同一个文件将会有多个不同命名的复制对象。那么我们的这个例子就是来完成这个工作的，它将所有的主机名统一设置为 varnish-software.com

- req.hash_always_miss：该变量用于设定是否强制一个请求总是不会在缓存中被击中。如果设置为 true，那么 Varnish 将会忽略任何在缓存中存在的对象，而总是从后台服务器中获取（也可以称为重新获取）。
- req.hash_ignore_busy：该变量用于设置任何在缓存中被频繁查询的对象，都将忽略其查询。如果有两个服务器用于查询内容，那么可以这么做，以避免潜在的死锁。

下列变量在准备后端请求的时候有效（要么是缓存没有击中，要么是 pass 或者 pipe）：

- bereq.request：用于设定请求的类型（例如“GET”，HEAD“）。
- bereq.url：指定客户端请求的 URL。
- bereq.proto：该变量用于指定与服务器端通信所使用的 HTTP 协议版本。
- bereq.http.header：相应的 HTTP 头。
- bereq.connect_timeout：设定与后台服务器连接超时，单位为秒。
- bereq.first_byte_timeout：等待第一个字节从后台服务器返回的时间设置，超过这个时间则视为超时，单位为秒。该变量在 pipe 模式下无效。
- bereq.between_bytes_timeout：该变量用于设置从后台服务器接收传输数据时，两个字节之间的最长时间间隔，超过这个间隔，则会被认为超时，单位为秒。该变量在 pipe 模式下无效。

下列变量在请求的对象从后端服务器检索到之后，且在该对象进入缓存之前才会有效，换句话说，它们只在 vcl_fetch 中有效：

- beresp.proto：当对象被检索到的时候，使用的 HTTP 协议版本。

- beresp.status：由服务器返回的 HTTP 状态代码。
- beresp.response：由服务器返回的状态信息。
- beresp.cacheable：如果客户端的请求导致了一个可缓存的响应，那么该变量的结果将会为真（true）。如果 HTTP 状态代码 200、203、300、301、302、404 或者 410，以及在 vcl_recv 中没有调用 pass，那么响应被认为是可缓存的。然而如果 TTL 和宽限时间（grace time）都为 0，那么 beresp.cacheable 就是 0，beresp.cacheable 是可以设置的。
- beresp.ttl：该变量是对象剩余的生存期，单位为秒。beresp.ttl 是可以设置的。看一下这个例子：

```
sub vcl fetch {
if (req.url ~ "^/logo_pic/") {
set beresp.ttl = 7d;
}
}
```

该例子通过设置 beresp.ttl 将 logo_pic 目录下的文件存放 7 天。

下列变量（大部分是只读变量），在对象进入缓存后才有效，就是当对象已经在缓存中的时候，典型的应用是 vcl_hit 和 vcl_deliver：

- obj.proto：当对象被检索到的时候，使用的是 HTTP 协议版本。
- obj.status：服务器返回的状态代码。
- obj.response：服务器返回的状态信息。
- obj.ttl：对象的剩余生存时间，单位为秒，obj.ttl 是可以设置的。
- obj.lastuse：指从对象被最后一次请求到现在过去的时间，单位为秒。
- obj.hits；对象被投递的次数，即被命中的次数，如果这个值为 0，那么表示缓存没有被命中，即缓存出错。

下列变量在确定一个对象的散列键时有效：

req.hash：在缓存中，哈希键被用于指向一个对象，无论是从缓存中读取或者是写入对象时，代表对象的哈希键都会被使用。

下列变量在为客户端准备响应时才有效：

- resp.proto：提供响应时所使用的 HTTP 协议版本。
- resp.status：返回的 HTTP 状态代码。
- resp.response：返回的 HTTP 状态信息。
- resp.http.header：相应的 HTTP 头。

通过使用 set 关键字可以为变量指定值：

```
sub vcl_recv {
  # Normalize the Host: header
  if (req.http.host ~ "^(www.)?example.com$") {
set req.http.host = "www.example.com";
  }
```

```
}
```

通过使用 remove 关键字可以完全移除 HTTP 头：

```
sub vcl fetch {
  # Don't cache cookies
  remove beresp.http.Set-Cookie;
}
```

57.13.5　ACL 指令

下面是 ACL 中的指令，也可以称做是关键字。它们执行着不同的操作。

1. log

可以通过 log 指令来记录任意的字符串到共享内存日志。

2. include

任何 VCL 的内容可以插入到其他的 VCL 文件中，插入的方法是使用 include 指令，可以在代码的任何位置插入，插入格式是在 include 指令之后跟随文件名字，文件名字由双引号括起。例如：

```
# in file "main.vcl"
include "backends.vcl";
include "purge.vcl
```

3. backend

后台服务器声明是使用 backend 指令来完成的，backend 指令用于宣布创建并且初始化一个命名的后台对象（backend object），看下面的配置：

```
backend www {
  .host = "www.example.com";
  .port = "http";
}
```

在这个配置中，指令 backend 之后的 www 就是命名的后台对象。

我们通过 backend 指令定义的这个 www 对象，在后面的请求配置中将会被使用。例如：

```
if （req.http.host ~ "^（www.）?example.com$"） {
  set req.backend = www;
}
```

指令 backend 除了.host 和.port 参数之外，还有以下参数。

- .max_connections：为了避免后台服务器过载，通过该参数可以设置同时对后台服务器连接的最大数量。
- .connect_timeout：该参数用于设定等待后台服务器连接的时间。
- .first_byte_timeout：该参数用于设定从后台服务器返回的第一个字节的时间。
- .between_bytes_timeout：该参数用于设定所收到字节的每个字节之间的间隔。

例如：

```
backend www {
  .host = "www.example.com";
  .port = "http";
  .connect timeout = 1s;
  .first byte timeout = 5s;
  .between bytes timeout = 2s;
}
```

4. directors

指令 director 用于定义一个 backend 服务器组，它能够实现冗余服务器集群。它的基本职能是让 Varnish 在发生故障即宕掉后，在这些服务器之间选择一个其他的 backend 使用。

对于 director 指令还有不同的类型可以使用，不同的 director 类型使用了不同的算法来选择 backend 使用。

配置 director 的方法类似如下格式：

```
director b2 random {
  .retries = 5;
  {
// We can refer to named backends
.backend = b1;
.weight  = 7;
  }
  {
// Or define them inline
.backend  = {
  .host = "fs2";
}
  .weight = 3;
  }
}
```

下面介绍 director 的五种类型：random director、round-robin director、client director、hash director 和 DNS director。

random director

使用 random director 指令时，每一条 random director 指令都需要使用.retries 选项，它的功能在于指定尝试后服务器的次数。对于每一个 random director 指令，.retries 对每个 backend 的定义是相同的。

也有针对每一个 backend 设置的选项：weight 定义了发送请求到后端服务器流量的百分比，例如，.“weight = 8”为 80%。

例如：

```
director backend-c random {
 .retries = 6;
```

```
{
/* backend servers */
.backend= b1;
.weight = 8;
}
{
/* another host */
.backend= {
.host = "192.168.189.23";
}
.weight = 2;
}
}
```

round-robin director

round-robin director 轮询，没有可设置的选项。

```
director backend-d round-robin {
{ .backend = server-w; }
{ .backend = { .host = "w3.xx.com"; .port = "80"; } }
}
```

client director

使用 client director 指令时，varnishd 将会依据 client 的身份（identity）来选择使用。我们可以使用 VCL 变量 client.identity，该变量通过使用会话 cookie 或者其他类似的值来标识客户端。

> 注意，在 2.1 版本中，变量 client.identity 已不再有效，director 将会使用 client.ip 将客户请求传递到 backend。

client director 有一个 retries 选项，它用于设定查找次数以便发现健康的后端服务器。

```
backend t1 {
.host = "192.168.9.15";
.port = "8081";
}
backend t2 {
.host = "192.168.9.16";
.port = "8082";
}
director backend-t client {
{
    .backend = t1;
    .weight = 1;
}
{
    .backend = t2;
```

```
        .weight = 1;
    }
    }
```

hash director

hash director 会根据 URL 的哈希值选择一个后台服务器。这种方式的可用之处是，如果在其他 Varnish 缓存或者是 Web 加速器之前使用了 Varnish 实现负载均衡，那么使用这种 director 将不会出现不同的缓存服务器放置同一份对象。

DNS director

DNS director 可以在三种不同的方式下使用，即使用 random、round-robin director 或者.list：

```
director backend-DNS dns {
.list = {
.host_header = "www.xx.com";
.port = "80";
.connect_timeout = 0.4;
"192.168.9.0"/24;
"192.168.10.0"/24;
}
.ttl = 5m;
.suffix = "internal.xx.net";
}
```

使用这种配置，我们一次性指定了（256-2）*2=508 backend，并且都使用 80 端口，连接超时为 0.4s。在.list 中，选项的定义一定要在 IP 列表之前定义。

- .ttl：定义了 DNS 查询缓存的时间。
- .suffix：在上面的例子中，查询之前会把“internal.xx.net”添加在客户端所支持的主机头（Host header）上。

所有这些选项为可选。

5. probe（Backend 探测器）

使用 backend 探测器的原因在于测试后端的服务器是否健康，也可以通过使用 req.backend.healthy 来返回健康状态。probe 中可设置的选项如下。

- .url：默认请求的 URL 格式。（在同一个 probe 中，我们必须在.url 和 .request 这两者之间选择）
- .request：指定完整的请求。在 HTTP 请求发送到后台服务器中，每一个被指定的字符串将会以单独的一行出现。（在同一个 probe 中，我们必须在.url 和 .request 这两者之间选择）

例如：

```
.request =
   "GET /test.jpg HTTP/1.1"
   "Host: 192.168.9.10 "
   "Connection: close"
```

```
    "Accept-Encoding: foo/bar" ;
```

- .window：设定最新检测次数。
- .threshold：用于设定成功尝试指定的次数后就被认为是健康的服务器。
- .initial：该参数用于设定当 Varnish 启动时多少个 probe 被看做是正常值，如果没有指定它，默认值同 threshold 一样。

例如：

这是一个带有 probe 的 backend 设置：

```
backend www {
  .host = "www.xx.com";
  .port = "http";
  .probe = {
.url = "/test.jpg";
.timeout = 0.3s;
.window = 8;
.threshold = 3;
.initial = 3;
  }
}
```

也可能指定原始的 HTTP 请求：

```
backend www {
  .host = "www.xx.com";
  .port = "http";
  .probe = {
# NB: \r\n automatically inserted after each string!
.request =
  "GET / HTTP/1.1"
  "Host: www.foo.bar"
  "Connection: close";
  }
}
```

6. ACL

acl，访问控制列表，它用于创建和初始化一个访问控制列表，在后面的配置中匹配客户端 IP 地址时使用。

```
acl local {
  "localhost"; // myself
  "192.0.2.0"/24;  // and everyone on the local network
  ! "192.0.2.23";  // except for the dialin router
}
```

在上面的配置中，acl 之后的 local 就是在后面的配置中使用的访问控制列表名称。如果在 ACL 条目中指定了 Varnish 服务器不能够解析的主机名称，那么它会匹配任意地址，这一点要注

意。在配置中使用了“！”号，它表示否定，即任何与该条目匹配的地址都将会被弹回，也就是拒绝访问。如果一个条目被圆括号括起，那么该条目将会被简单地忽略。

看下面的配置：

```
if (client.ip ~ local) {
  return (pipe);
}
```

在这个配置中，通过客户端 IP 地址匹配 ACL 中的 IP 地址来决定是否可以访问资源，在这里只需简单地使用匹配操作符“~”。

7. set、unset 和 remove

可以使用 set 关键字来设置任意的 HTTP 头，也可以使用 unset 或者 remove 来移除 HTTP 头，unset 和 remove 是同义词。

看下面的这个例子：

```
if (req.http.Accept-Encoding) {
if (req.url ~ "\.(jpg|png|gif|gz|tgz|bz2|tbz|mp3|ogg)$") {
# No point in compressing these
remove req.http.Accept-Encoding;
} elsif (req.http.Accept-Encoding ~ "gzip") {
set req.http.Accept-Encoding = "gzip";
} elsif (req.http.Accept-Encoding ~ "deflate") {
set req.http.Accept-Encoding = "deflate";
} else {
# unkown algorithm
remove req.http.Accept-Encoding;
}
}
```

在这个例子中，我们针对 Vary 头来讲述，Vary 头是由 Web 服务器发送的。当一个服务器发布了一个“Vary：Accept-Encoding”，这就是告诉 Varnish 要为每一个来自不同客户端的 Accept-Encoding 缓存一个单独的版本。因此，如果一个客户端仅接受 gzip 编码，那么 Varnish 将不会提供以 deflate 编码的页面。

问题出在 Accept-Encoding 字段包括了太多不同的编码。如果一个浏览器发送：

```
Accept-Encodign: gzip,deflate
```

而另一个浏览器发送：

```
Accept-Encoding:: deflate,gzip
```

由于不同的 Accept-Encoding 头，Varnish 将会保持两个不同的变体页面。在缓存中存储应该尽可能地减少变体，因此，我们在这个例子使得 Accept-Encoding 正常化。

我们针对图片、音频及压缩包都将移除 Accept-Encoding，而 Accept-Encoding 匹配 gzip 的情况则设置为 gzip；Accept-Encoding 匹配 deflate 的情况则设置为 deflate；如果与这些都不匹配的情况，那么设置为移除 Accept-Encoding。

另外，有些应用程序或者应用服务器在内容中会发送 Vary: User-Agent。同样，这将会命令 Varnish 为每一个变异 User-Agent 缓存一份复制。因此，如果确实是需要基于 User-Agent 的 Vary，那么要正常化它的头，否则，命中率将会下降。可以使用上面的例子作为模板来修改实现。

关于 Vary 头，我们再提供 Nginx 了解一下，在 Nginx 的配置中，如果不使用“ gzip_vary on;”，那么无疑会提高缓存的命中率。

57.13.6 Varnish 的函数

下列是 Varnish 内置的函数：

hash_data（str）

将添加的字符串进行哈希处理后再输出。在 default.vcl 文件中，hash_data（）被用于 host 和 URL 请求。

例如：

```
sub vcl_hash {
hash_data (req.url) ;
 if  (req.http.host)  {
hash_data (req.http.host) ;
 } else {
 hash_data (server.ip) ;
 }
 return  (hash) ;
}
```

regsub（str,regex,sub）

该函数用于替换第一次匹配 regex 表达式返回的 str。在 sub 中，0（也可以被写为&）被整个匹配的字符串替代，n 则表示在匹配的字符串中仅匹配前 n 个。

下面的代码可以实现忽略掉查询参数而仅缓存对象本身，这个正则表达式将会移除查询参数：

```
sub vcl recv {
set req.url = regsub (req.url, "\?.*", "") ;
}
```

看下面的一个例子：

```
if  (req.http.host ~ "^ (www.) ?xx.com")  {
  set req.url=regsub (req.url,"^","/Vhostbase/xx.com:80/Sites/
     xx.com/Root") ;
}
```

在这个例子中我们通过函数 regsub（）实现了在请求的 URL 到后端服务器之前将其重定向。

- regsuball（str，regex，sub）：替换所有发现的目标。
- purge_url（regex）：清空缓存中所有与正则表达式匹配的内容。

57.13.7 子程序

1. 自定义子程序

在 Varnish 的配置文件中，我们可以自定义子程序。子程序被用于组织代码以便实现可读性和可重用性。例如：

```
sub pipe if local {
if (client.ip ~ local) {
  return (pipe);
}
}
```

定义子程序通过 sub 来完成，在 VCL 中，子程序不带参数，也没有返回值。调用子程序可以使用关键字 call，然后跟随使用 sub 定义的子程序名称。

例如：

```
call pipe_if_local;
```

2. 内定义子程序

在 Varnish 中，内定义了一些特殊的子程序，它们被挂接到 Varnish 的工作流程中。这些子程序可以检查和熟练控制 HTTP 头（header），以及每一个程序的其他方面，并在一定程度上决定应该如何处理该请求。每一个子请求的终结是通过调用所期望行为的关键字而结束。

在 VCL 中内定义了以下的子程序。

vcl_recv

该函数将会在请求开始时被调用，在完整的请求被收到后，请求就会被解析，然后该函数再决定是否为该请求提供服务，以及如何响应、使用哪一个 backend。

vcl_recv 子程序能够通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- Pass：切换到 pass 模式。控制权最终将传递给 vcl_pass。
- Pipe：切换到 pipe 模式。控制权最终将传递给 vcl_pipe。
- Lookup：在缓存中查询请求的对象。控制权最终将传递给 vcl_hit 或者 vcl_miss，具体是哪一个，则依赖于查询的对象是否被击中。

在 vcl_recv 中，我们可以修改请求，例如，添加或者删除请求的头信息，还可以修改 cookies，等等。

格式：

```
# sub vcl_recv {
# if (req.restarts == 0) {
#  if (req.http.x-forwarded-for) {
#  set req.http.X-Forwarded-For =
#  req.http.X-Forwarded-For + ", " + client.ip;
#  } else {
#  set req.http.X-Forwarded-For = client.ip;
```

```
#   }
# }
# if (req.request != "GET" &&
#   req.request != "HEAD" &&
#   req.request != "PUT" &&
#   req.request != "POST" &&
#   req.request != "TRACE" &&
#   req.request != "OPTIONS" &&
#   req.request != "DELETE") {
# /* Non-RFC2616 or CONNECT which is weird. */
# return (pipe);
# }
# if (req.request != "GET" && req.request != "HEAD") {
# /* We only deal with GET and HEAD by default */
# return (pass);
# }
# if (req.http.Authorization || req.http.Cookie) {
# /* Not cacheable by default */
# return (pass);
# }
# return (lookup);
# }
```

vcl_pipe

该函数在请求进入 pipe 模式的时候会被调用。在这种模式下，请求将被传递到后端服务器，一旦连接，在连接关闭之前，无论是客户端还是与之相对应的后端服务器的数据都会在 pass 模式中。就是说，进入 pipe 模式时，请求被直接发送到后端服务器，后端和客户端之间后继数据将不再进行处理，只是简单传递，直到一方关闭连接。

vcl_pipe 子程序同样通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- pipe：继续使用 pipe 模式。

格式：

```
# sub vcl_pipe {
# # Note that only the first request to the backend will have
# # X-Forwarded-For set.  If you use X-Forwarded-For and want to
# # have it set for all requests, make sure to have:
# # set bereq.http.connection = "close";
# # here.  It is not set by default as it might break some broken web
# # applications, like IIS with NTLM authentication.
# return (pipe);
# }
```

vcl_pass

进入 pass 模式调用时，进入该模式后将会使用 vcl_pass 子程序。在这种模式下，请求被传

递到后台服务器，并且后台服务器的响应将会传递到客户端，但是在这种情况下，响应的内容并不会进入缓存，同一连接的子请求将会被正常处理。

vcl_pass 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- pass：切换到 pass 模式。控制权最终将传递给 vcl_pass。
- restart：重新启动事务处理。增加了重启计数器。如果重启的数值大于 max_restarts，那么将会发出错误。

格式：

```
# sub vcl pass {
# return （pass）;
# }
```

vcl_hash

该函数的功能在于调用 hash_data()为数据添加哈希值。vcl_hash 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

hash：继续进入 hash 模式处理。

这是 2.1 版本中的格式：

```
# sub vcl_hash {
# set req.hash += req.url;
# if （req.http.host） {
# set req.hash += req.http.host;
# } else {
# set req.hash += server.ip;
# }
# return （hash）;
# }
```

这是 3.0 版本中的格式：

```
# sub vcl hash {
# hash_data（req.url）;
# if （req.http.host） {
# hash_data（req.http.host）;
# } else {
# hash_data（server.ip）;
# }
# return （hash）;
# }
```

vcl_hit

在缓存查询后，如果文档对象在缓存中被找到，那么调用 vcl_hit。vcl_hit 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- deliver：将被缓存的对象交付给客户端。控制权最终将会传递给 vcl_deliver。
- error code [reason]：给客户端返回指定的代码，并且放弃该请求。

- pass：切换到 pass 模式。控制权最终将会传递给 vcl_pass。
- restart：重新启动事务处理。增加了重启计数器。如果重启的数值大于 max_restarts，那么将会发出错误。

这是 2.1 版本中的格式：

```
# sub vcl_hit {
# if （!obj.cacheable） {
# return （pass）;
# }
# return （deliver）;
# }
```

这是 3.0 版本中的格式：

```
# sub vcl_hit {
# return （deliver）;
# }
```

vcl_miss

在缓存中查询（lookup）后，如果查询的文档对象在缓存中没有被找到，那么就会调用函数 vcl_miss。它的目的在于决定是否尝试从后台服务器查找文档（请求的内容），以及使用哪一个后台服务器。

vcl_miss 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- pass：切换到 pass 模式。控制权最终将会传递给 vcl_pass。
- fetch：从后台服务器查找请求的对象。控制权最终将会传递给 vcl_fetch。

格式：

```
# sub vcl miss {
# return （fetch）;
# }
```

vcl_fetch

当一个文档成功地从后台服务器检索到后就会调用该函数。

vcl_fetch 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- deliver：可能会将对象插入缓存，然后再将它转交给客户端。控制权最终将会传递给 vcl_deliver。
- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- esi：ESI 进程处理刚刚获取的文件。
- pass：切换到 pass 模式。控制权最终将会传递给 vcl_pass。
- restart：重新启动事务处理。增加了重启计数器。如果重启的数值大于 max_restarts，那么将会发出错误。

这是 2.1 版本中的格式：

```
# sub vcl fetch {
# if （!beresp.cacheable） {
```

```
# return (pass);
# }
# if (beresp.http.Set-Cookie) {
# return (pass);
# }
# return (deliver);
# }
```

这是 3.0 版本中的格式：

```
# sub vcl_fetch {
# if (beresp.ttl <= 0s ||
# beresp.http.Set-Cookie ||
# beresp.http.Vary == "*") {
#   /*
#* Mark as "Hit-For-Pass" for the next 2 minutes
#*/
#   set beresp.ttl = 120 s;
#   return (hit_for_pass);
# }
# return (deliver);
# }
```

vcl_deliver

该函数将会在缓存对象被投递到客户端之前调用。

vcl_deliver 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- deliver：将对象交付给客户端。
- error code [reason]：给客户端返回指定的代码，并且放弃该请求。
- restart：重新启动事务处理。增加了重启计数器。如果重启的数值大于 max_restarts，那么将会发出错误。

格式：

```
# sub vcl deliver {
# return (deliver);
# }
```

vcl_error

当命中一个错误时会调用该函数，错误的原因是因为后端服务器明确的或者是含蓄的也有可能是内部的错误。通过该函数可以来定制一个错误页面。

vcl_error 子程序可以通过调用 return()来终结。对于 return()来说，可选择的关键字如下。

- deliver：将对象交付给客户端。
- restart：重新启动事务处理。增加了重启计数器。如果重启的数值大于 max_restarts，那么将会发出错误。

这是 2.1 版本中的格式：

```
# sub vcl_error {
# set obj.http.Content-Type = "text/html; charset=utf-8";
# synthetic {"
# <?xml version="1.0" encoding="utf-8"?>
# <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
#  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
# <html>
#   <head>
# <title>"} obj.status " " obj.response {"</title>
#   </head>
#   <body>
# <h1>Error "} obj.status " " obj.response {"</h1>
# <p>"} obj.response {"</p>
# <h3>Guru Meditation:</h3>
# <p>XID: "} req.xid {"</p>
# <hr>
# <p>Varnish cache server</p>
#   </body>
# </html>
# "};
# return (deliver);
# }
```

这是 3.0 版本中的格式：

```
# sub vcl_error {
# set obj.http.Content-Type = "text/html; charset=utf-8";
# set obj.http.Retry-After = "5";
# synthetic {"
# <?xml version="1.0" encoding="utf-8"?>
# <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
#  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
# <html>
#   <head>
# <title>"} + obj.status + " " + obj.response + {"</title>
#   </head>
#   <body>
# <h1>Error "} + obj.status + " " + obj.response + {"</h1>
# <p>"} + obj.response + {"</p>
# <h3>Guru Meditation:</h3>
# <p>XID: "} + req.xid + {"</p>
# <hr>
# <p>Varnish cache server</p>
#   </body>
# </html>
```

```
    # "};
    # return (deliver);
    # }
```

如果这些子程序中有未定义（包括定义不明确的情况）或者是没有达成处理决定而终止，那么控制权将会被转交给系统默认的子程序。

3. 多个子程序的处理

如果在 VCL 中定义了多个同名的子程序，那么它们将被按照在源代码中出现的顺序连接起来。

例如：

```
# in file "main.vcl"
include "backends.vcl";
include "purge.vcl";

# in file "backends.vcl"
sub vcl_recv {
  if (req.http.host ~ "example.com") {
set req.backend = foo;
  } elsif (req.http.host ~ "example.org") {
set req.backend = bar;
  }
}

# in file "purge.vcl"
sub vcl_recv {
  if (client.ip ~ admin_network) {
if (req.http.Cache-Control ~ "no-cache") {
  purge_url(req.url);
}
  }
}
```

在这个例子中我们看到，在配置“main.vcl”中，包含了两个 vcl 文件，而每个文件中都定义了一个 vcl_recv 函数，因此，在对“main.vcl”编译时，它们（这两个 vcl_recv 函数）将会被按照在“main.vcl”中出现的顺序连接起来。

对于 Varnish 内建的默认函数，也是通过这种方法隐含添加处理。

57.13.8 ESI

ESI 标识语言（Edge Side Includes）是一种简单标记语言，用于定义动态内容网页的构成成分以及互联网边缘的网站应用程序的分发。

ESI 标识语言提供了一种内容管理机制，这些内容包括应用程序服务器解决方案、内容管理系统，以及内容传送网络。因此，ESI 让公司能够进行网络应用的一次性开发，并在安装应用程序时，选择好内容管理系统、应用服务器或内容分发网络的配置时间，这样就可降低复杂性、开发时间和配置成本。

ESI 标识语言的公开标准技术规范是由 Akamai、ATG、BEA Systems、Circadence、Digital Island、IBM、Interwoven、Oracle 和 Vignette 等公司共同编写的。

—— 本内容来自互联网

在大多数 Web 站点中，许多内容在多个页面中共享，如果每个页面都重新产生一次这些共享的页面，那么势必会造成资源的浪费。ESI 设法通过地址定位的方法来允许我们对每一个嵌入的片段个别地设置，以便由我们决定缓存策略。

在 Varnish 中，我们仅能够执行 ESI 的一小部分子集，在现有的 Varnish 版本（2.15 和 3.0.0）中有以下三种 ESI 使用方法。

- esi：include
- esi：remove
- <!–esi …–>

举例：esi include

我们看以下这个例子，这个简单的 CGI 脚本用于输出当前的日期：

```
#!/bin/sh

echo 'Content-type: text/html'
echo ''
date "+%Y-%m-%d %H:%M"
```

在 HTML 文件中嵌入 ESI 片段：

```
<HTML>
<BODY>
The time is: <esi:include src="/cgi-bin/date.cgi"/>
at this very moment.
</BODY>
</HTML>
```

但是如果想让 ESI 工作，那么还要在 VCL 中激活 ESI，例如，在 VCL 文件中配置类似代码：

```
sub vcl_fetch {
if (req.url == "/test.html") {
   set beresp.do_esi = true; /* Do ESI processing   */
   set obj.ttl = 24 h;   /* Sets the TTL on the HTML above  */
} elseif (req.url == "/cgi-bin/date.cgi") {
   set obj.ttl = 1m; /* Sets a one minute TTL on*/
 /*  the included object*/
}
}
```

举例：esi remove

remove 关键字允许我们移除输出内容，当 ESI 不再可用时，我们使用它作为一个勉强的后退，就是在正确的内容无效时避免输出不必要的内容：

```
<esi:include src="http://www.example.com/ad.html"/>
<esi:remove>
  <a href="http://www.example.com">www.example.com</a>
</esi:remove>
```

举例：<!-esi ... ->

这是一个特殊的组成部分，它允许 HTML 标记 ESI 不进行处理，如果进行 ESI 处理，那么 ESI 将会移除开始（<!-esi）和结束（->）处的标记；如果不做 ESI 处理，那么它仍然会被保留，会作为 HTML/XML 的一个注释标记。例如：

```
<!--esi
<p>Warning: ESI Disabled!</p>
</p>  -->
```

这种方式保证了如果 ESI 不执行，也将不会妨碍最终 HTML 的解释执行。

57.14 grace 模式和 saint 模式

如果后端服务器需要较长的时间来生成一个对象，可能就会有线程堆的风险了。为了阻止这种现象的发生，可以使用 grace 方式，这种方式允许在后端服务器生成新的对象期间，Varnish 可以提供该对象的过期版本。

下面的 VCL 代码将会使得 Varnish 提供过期的对象，所有的对象将在生存期满后保存 20 分钟或者是生成新的对象。

```
sub vcl_recv {
  set req.grace = 20m;
}
sub vcl fetch {
  set beresp.grace = 20m;
}
```

saint 模式类似于 grace 模式，依赖于同样的基础设施，但是功能却不同。我们可以向 vcl_fetch 添加 VCL 代码看看后台服务器是否会返回我们期望的响应，如果发现响应不是期望的响应，那么你可以通过设置 beresp.saintmode 来做一个时间限制并且调用 restart，然后 Varnish 会重新尝试其他的后台服务器以便再次获取该对象。

```
sub vcl_fetch {
 if （beresp.status == 500） {
 set beresp.saintmode = 10s;
 restart;
 }
 set beresp.grace = 45m;
}
```

这个例子是从 wiki 上演绎而来的，如果 director 发生问题，那么 saintmode 是另外一种告知的方法。

在这里，我们设置 beresp.saintmode 的值为 10 秒，在这样的配置下，Varnish 在 10 秒内将不会访问后端服务器的这个 url。

如果有一个备用列表，当重新执行此请求时，你有其他的后端有能力提供此服务内容，Varnish 会尝试请求它们，当你没有可用的后端服务器，Varnish 将使用它过期的 cache 提供服务内容。

57.14.1 grace 模式

当多个客户端请求相同的页面时，Varnish 将只发送一个请求到后端服务器，而暂时搁置（或者叫挂起）其他的客户端请求等待返回的结果，当结果返回后，将结果从后台复制到每一个请求，这些都是由 Varnish 自动完成。

这样做的好处是在 Varnish 缓存内没有该 url，并且有上千个线程来请求同一个 url 时，Varnish 仅取一次源，降低了消耗并且提高了速度。

如果对于一个每秒钟提供成千上万点击请求的服务器来说，那么等待请求的队列将会变得巨大。对于这种模式会有以下两个潜在的问题：一是异常大的群集问题——突然地释放成千上万的线程来发送服务器内容使负载出奇得高；二是没有人喜欢等待，要解决这个问题，我们可以通过延长对象的生存期，以便提供缓存中过期的内容。

因此，为了提供过期的内容，我们必须有一些内容以便提供服务，因此，按照下面的设置，它的意思是当缓存的对象超过 TTL 生存期后，30 分钟内将不会删除：

```
sub vcl_fetch {
  set beresp.grace = 30m;
}
```

只有这个配置还不能提供过期的对象，为了能够使得 Varnish 真正提供过期的对象，我们必须为请求设置这个选项：

```
sub vcl recv {
  set req.grace = 15s;
}
```

这个配置的意义在于：在缓存过期后的 15 秒内，使用旧的内容提供服务。

在前面的设置中，我们将缓存中的对象保留了 30 分钟，对于这个你可能会感到惊讶。当然，如果开启了健康检查，我们可以通过健康检查来设置保存的时间：

```
if (! req.backend.healthy) {
   set req.grace = 5m;
} else {
   set req.grace = 15s;
}
```

总而言之，grace 模式解决了以下两个问题：

- 它能够提供陈旧的内容，以避免请求堆积；
- 如果后端服务器出现不健康情况，那么它能够提供过期的内容。

57.14.2 saint 模式

有的时候服务器会变得很古怪，它们可能会抛出各种随机的错误。在这种情况下，你需要通知 Varnish 以比较优雅的方式来尝试处理这些错误。因此，我们就可以使用前面提到的 saint 模式。我们看一下在 VCL 中如何设置：

```
sub vcl fetch {
  if （beresp.status == 500） {
set beresp.saintmode = 10s;
restart;
  }
  set beresp.grace = 5m;
}
```

在这个例子中，我们将 beresp.saintmode 设置为 10 秒，那么对于这个 URL 的请求 Varnish 在 10 内将不会访问服务器，它的做法差不多类似于黑名单。如果有其他后端服务器有能力提供这些内容，那么也可以执行一个 restart，然后 Varnish 将会尝试这些服务器。当我们没有可用的后端服务器时，那么 Varnish 将会提供缓存中的过期内容。

57.14.3 grace 模式和 saint 模式的局限性

当对象在被获取的过程中请求发生失败时，那么我们得到的将是 vcl_error 抛出的错误。vcl_error 仅能够访问一组有限的数据，因此，对于它在这里就不要开启 saint 或 grace 模式。这种情况可能会在将来的版本中解决，但是可用采取变通的方法：

- 声明一个后台服务器总是有问题的；
- 在 vcl_error 中设置魔术标记；
- 重新启动处理过程；
- 注意在 vcl_recv 中的魔术标记要在后台服务器中做相应的应用；

Varnish 将会提供任何过期的可见的数据。